SCREW THEORY BASED PATH TRACKING TECHNIQUES FOR AUTONOMOUS ROBOTS AND MANIPULATORS

By

WAHEED A. ABBASI

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2001

Dedicated to my parents.

ACKNOWLEDGMENTS

I would like to express my thanks and appreciation to the members of my supervisory committee: Dr. Carl Crane, Dr. Joseph Duffy, Dr. John Ziegert, Dr. John Schueller, and Dr. Michael Nechyba for their support and guidance. I extend special thanks to committee chairman Dr. Crane, for his guidance, patronage, and support throughout my tenure.

I thank Dr. José Rico Martinez for his invaluable input and his friendship. I also thank my friends and colleagues in CIMAR who provided a very cordial and friendly work environment.

I give my heartfelt gratitude to my parents, whose love, dedication, and encouragement made whatever I have done possible. Finally, I give special thanks to my wife, Uzma Abbasi, who provided continuous encouragement and the inspiration needed to finish this dissertation.

TABLE OF CONTENTS

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

SCREW THEORY BASED PATH TRACKING TECHNIQUES FOR AUTONOMOUS ROBOTS AND MANIPULATORS

By

WAHEED A. ABBASI

August 2001

Chairman: Dr. Carl D. Crane III
Major Department: Mechanical Engineering

The Theory of Screws, invented by Sir Robert Ball, has been used in the mechanisms area for over three decades. Engineers and scientists at the Center for Intelligent Machines & Robotics (CIMAR) have successfully used the Theory for a variety of applications including analysis of serial and parallel mechanisms, tensegrity structures, and vibration analysis. The theory provides for an elegant and geometrically meaningful way for conducting such analyses.

Following a planned path is an essential element in most applications of robots and manipulators. Following a path necessitates the determination of how accurately the path is followed. This is generally known as path tracking. This research is aimed at applying the Theory of Screws to the path tracking problem. The research forays into developing a path tracking methodology for autonomous vehicles, serial robots, and parallel manipulators based on Screw Theory.

The path tracking problem is addressed by developing mathematical basis for generating the tracking commands. The techniques are based on whether the tracking needs to be done at the position level or at the velocity level, i.e., whether the trajectory has to be continuous at the position level or at the velocity level. The basic models are developed including the resultant of combining a velocity and an acceleration command screw. The developed path tracking methodology is applied to autonomous robots and serial manipulators. A separate methodology is developed for parallel mechanisms. A technique is devised that satisfies the motion criteria for the desired position, velocity, and acceleration commands. The technique provides the connector displacements, velocities, accelerations, and forces for a prescribed motion of the platform. The methodology can be used as a mechanism performance evaluation tool. Simulations are performed to validate the results.

A dynamic model of a 3R manipulator is developed to test the path tracking algorithm with a control system. A PID controller is used for manipulator motor control. The control system is modeled with a time delay, which is varied to study its effects on system performance. The results show that the path tracking techniques function for the test manipulator within a range of time delays.

CHAPTER 1
INTRODUCTION

Motivation

The motivation for this work comes from the elegance with which the Theory of Screws has been used for the kinematic and dynamic analyses of robot manipulators. The initiative is to employ the Theory of Screws (Screw Theory) and attempt to develop a methodology for path tracking of robots.

Following a planned path is a fundamental function in most applications of robots, manipulators, machine tools, and autonomous vehicles[1]. In most cases, following a planned path is not only necessary but is critical in accomplishing the task; necessitating the determination of how accurately the path is followed. This is accomplished by what is known as path tracking.

Path tracking in general can be defined as following a prescribed path with possible additional constraints. These additional constraints are generally system or task dependent. For robot manipulators and autonomous vehicles, these constraints can be orientation, time, velocity, and acceleration. A cursory review of scientific literature in the area of path tracking shows that path tracking is in general either a sub-system in the overall control scheme of the robot or is embedded in the control system.

[1] The term robot is used as a generality; it encompasses manipulators (serial and parallel), machine tools, and autonomous vehicles.

Background

Work in the area of mechanisms (among other research areas) has been ongoing at the Center for Intelligent Machines and Robotics (CIMAR) for decades. Research has been conducted and published on mechanism analysis, ranging from direct (forward) and inverse (reverse) analysis of mechanisms to velocity, acceleration, and jerk analysis of mechanisms.

An area of particular thrust has been the design and development of parallel mechanisms for a variety of applications. Parallel platforms have been, and are being, investigated for applications ranging from passive feedback devices to milling machines.

The development of mechanical robotic systems requires the need for a control system to accomplish the prescribed task that it is designed for. The idea of using Screw Theory to accomplish this task arose from this need.

Research Objectives

The objective of this research is to investigate the use of Screw Theory as a tool for path tracking of robot manipulators. The intent is to develop a methodology that lays the framework for tracking control. Techniques will be developed for tracking control of planar serial manipulators (3R), parallel manipulators (6-6 spatial manipulator), and autonomous vehicles. These techniques are implemented in simulation and results presented to show the performance and viability of the techniques.

The material is arranged in the following order. Chapter 2 offers a brief introduction to the Theory of Screws and to the concepts that are used in this research. Chapter 3 reviews the current state of literature as it pertains to the control of robotic manipulators in general, and to path tracking in particular. Chapter 4 presents the approaches that have been developed and the simulations that have been done to illustrate and validate the methodologies. Chapter 5 presents an application of the path tracking methodology to a 3R Manipulator with a system model and application of control strategies. Chapter 6 summarizes the pertinent work that has been done, concludes the results, and provides an outline for future work.

CHAPTER 2
BACKGROUND ON THEORY OF SCREWS

Since the work presented in this manuscript derives from the Theory of Screws, a brief overview of the relevant topics in the theory are presented to facilitate understanding. This introduction is not intended to be comprehensive, just illustrative enough to familiarize the reader with the material that follows. After the introductory material, results pertaining to the velocity and acceleration analyses of mechanisms based on Duffy, Crane, and Rico's work are presented.

Screw theory has been used in kinematics for more than two centuries. Sir Robert Ball conducted the principal and the most comprehensive fundamental work in this area at the end of the 19th century. This work was published as *A Treatise on the Theory of Screws* in 1900 [Bal00]. Except for a few isolated contributions in the first decades of this century, screw theory lay dormant until it was rediscovered in the 1960s by Hunt and Phillips [Hun78, Phi84] who made very important advances in the field. Since then, the theory has been used extensively in kinematics: from the search of over-constrained linkages, grasping analysis, velocity and singularity analysis of serial manipulators, to force, velocity and acceleration analysis of parallel manipulators [Mar96].

Definitions and Basic Concepts

Ball defined a screw in a manner that was impervious to whether its usage was in kinematics or dynamics. A screw is a geometrical identity described as "a straight line with which a definite linear magnitude termed the pitch is associated." In defining the pitch, Ball considers a rigid body that needs to go from one position to a desired position. A certain axis can be found such that if the body is rotated around that axis for a determinate angle and translated parallel to the axis for a determinate distance, the desired movement can be achieved. The situation is analogous to a body attached to the nut of a uniform screw (in the ordinary sense of the word) which has an appropriate position in space, and an appropriate number of threads to the inch. Thus the pitch of a screw is defined to be the rectilinear distance through which the nut is translated parallel to the axis of the screw, while the nut is rotated through the angular unit of circular measure. The pitch is thus a linear magnitude. It follows from this definition that the rectilinear distance parallel to the axis of the screw through which the nut moves when rotated through a given angle is simply the product of the pitch of the screw and the circular measure of the angle.

Twist: A body is said to receive a twist about a screw when it is rotated uniformly about the screw, while it is translated uniformly parallel to the screw, through a distance equal to the product of the pitch and the circular measure of the angle of rotation. Whatever the movement of a rigid body, it is at every instant twisting about a screw. For the movement of the body when passing from one position to another adjacent position, is indistinguishable from the twist about an appropriately chosen screw by which the same displacement could be affected. The screw about which the body is twisting at any instant is termed the instantaneous screw.

A system of forces acting on a rigid body may be generally expressed by a certain force and a couple whose plane is perpendicular to the force [Poi06]. A wrench is defined as a force and a couple in a plane perpendicular to the force. A wrench is thus a force directed along the screw and a couple in a plane perpendicular to the screw, the moment of the couple being equal to the product of the force and the pitch of the screw. It follows from this that if the pitch is zero, the wrench reduces to a pure force along the axis of the screw. The Lie product is a product of two screws It is also known as the Motor Product or Dual Vector Product and is defined as

$$\begin{aligned}[\$_1 \quad \$_2] &= [(\boldsymbol{\omega}_1 ; \boldsymbol{\upsilon}_{01}) \quad (\boldsymbol{\omega}_2 ; \boldsymbol{\upsilon}_{02})] \\ &= (\boldsymbol{\omega}_1 \times \boldsymbol{\omega}_2 \, ; \boldsymbol{\omega}_1 \times \boldsymbol{\upsilon}_{02} - \boldsymbol{\omega}_2 \times \boldsymbol{\upsilon}_{01}) \end{aligned} \tag{2.1}$$

A more mathematical description of entities follows.

Consider two distinct points $\mathbf{r}_1(x_1,y_1,z_1)$ and $\mathbf{r}_2(x_2,y_2,z_2)$ in space. The join of these two distinct points determines a line (Figure 2.1). The vector S directed along the line can be written as

$$\mathbf{S} = (\mathbf{r}_2 - \mathbf{r}_1) \tag{2.2}$$

which may be expressed as

$$\mathbf{S} = L\hat{i} + M\hat{j} + N\hat{k} \tag{2.3}$$

where $L = x_2 - x_1 \quad, M = y_2 - y_1 \quad, N = z_2 - z_1$ are defined as the direction ratios. From Equation (2.3), the direction ratios are related to the distance $|S|$ by

$$L^2 + M^2 + N^2 = |\mathbf{S}|^2 \tag{2.4}$$

Often the entities L, M, and N are expressed as

$$L = \frac{x_2 - x_1}{|\mathbf{S}|} \; , M = \frac{y_2 - y_1}{|\mathbf{S}|} \, , N = \frac{z_2 - z_1}{|\mathbf{S}|} \tag{2.5}$$

which represent the unit direction ratios or the direction cosines of the line. In this case (2.4) reduces to

$$L^2 + M^2 + N^2 = 1. \tag{2.6}$$

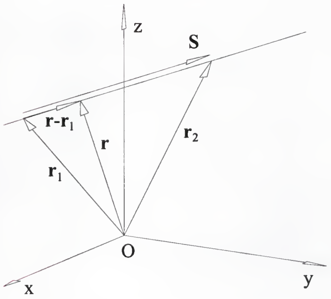


Figure 2.1: Two points determine a line.

In Figure 2.1, let **r** be a vector from the origin to any point on the line. The vector (**r**-**r**$_1$) is parallel to **S**, thus

$$(\mathbf{r} - \mathbf{r}_1) \times \mathbf{S} = 0. \tag{2.7}$$

This equation can be written as

$$\mathbf{r} \times \mathbf{S} = \mathbf{S}_0 \tag{2.8}$$

where

$$\mathbf{S}_0 = \mathbf{r}_1 \times \mathbf{S} \tag{2.9}$$

represents the moment of the line about the origin **O** and is origin dependent. Also, since the vectors S and S$_0$ are perpendicular they satisfy the orthogonality condition

$$\mathbf{S} \cdot \mathbf{S}_0 = 0. \tag{2.10}$$

A way of writing the coordinates of a line is in the form {S;S$_0$} and they are referred to as the Plücker coordinates of a line. The semi-colon is used to signify that the dimensions of |S| and |S$_0$| are different. The coordinates {S;S$_0$} are homogeneous since from (2.8) the coordinates $\{\lambda\mathbf{S};\lambda\mathbf{S}_0\}$, where λ is a non-zero scalar, determine the same line.

Expanding Equation (2.9) yields,

$$\mathbf{S}_0 = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ x_1 & y_1 & z_1 \\ L & M & N \end{vmatrix} \tag{2.11}$$

which can be written as

$$\mathbf{S}_0 = P\mathbf{i} + Q\mathbf{j} + R\mathbf{k}\,. \tag{2.12}$$

where

$$\left.\begin{aligned} P &= y_1N - z_1M\,, \\ Q &= z_1L - x_1N\,, \\ P &= x_1M - y_1L\,. \end{aligned}\right\}. \tag{2.13}$$

From (2.3) and (2.12) the orthogonality condition $\mathbf{S}.\mathbf{S}_0=0$ can be expressed in the form

$$LP + MQ + NR = 0 \tag{2.14}$$

If homogeneous coordinates of two points in space are provided as $(1;x_1,y_1,z_1)$ and $(1;x_2,y_2,z_2)$, the Plücker coordinates of the line (Figure 2.2) can be expressed by using the Grassmann method by taking six 2×2 determinants of the array

$$\begin{bmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \end{bmatrix} \tag{2.15}$$

as

$$\begin{gathered} L = \begin{vmatrix} 1 & x_1 \\ 1 & x_2 \end{vmatrix}, \quad M = \begin{vmatrix} 1 & y_1 \\ 1 & y_2 \end{vmatrix}, \quad N = \begin{vmatrix} 1 & z_1 \\ 1 & z_2 \end{vmatrix}, \\ P = \begin{vmatrix} y_1 & z_1 \\ y_2 & z_2 \end{vmatrix}, \quad Q = \begin{vmatrix} z_1 & x_1 \\ z_2 & x_2 \end{vmatrix}, \quad R = \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix}. \end{gathered} \tag{2.16}$$

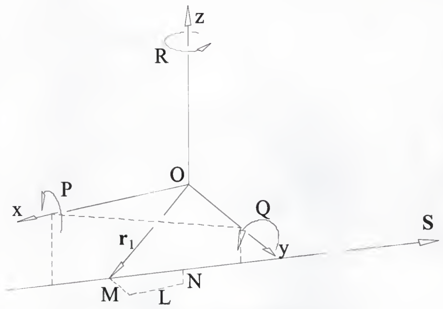


Figure 2.2: Plücker coordinates of a line.

Equations (2.2) and (2.9) can be used to determine the Plücker coordinates of a line when two points on the line are given. It is also important to determine a point on a line when the Plücker coordinates are given. If, for example, the coordinates of a line are given, {L,M,N;P,Q,R}, an infinite number of solutions exist for x,y,and z (corresponding to the infinite number of points on the line) [Cra99]. For the general case, any arbitrary value of x may be selected and the corresponding values for y and z may be determined as:

$$y = \frac{xM - R}{L} \quad , \quad z = \frac{xN + Q}{L} . \tag{2.17}$$

The distance of a line from the origin is determined by the length of the vector **p**, which originates at O and terminates at a point on the line such that the direction **p** is perpendicular to the direction of the line, **S**. The distance of the line from the origin is determined as

$$|\mathbf{p}| = \frac{|\mathbf{S}_0|}{|\mathbf{S}|}. \tag{2.18}$$

When $\mathbf{S}_0=0$, $|\mathbf{p}|=0$ and the line passes through the origin and its coordinates are $\{\mathbf{S};\mathbf{0}\}$. When $\mathbf{S}=\mathbf{0}$, $|\mathbf{p}|=\infty$, the line is a line at infinity and its coordinates are $\{\mathbf{0};\mathbf{S}_0\}$.

A line can be defined by the join of two points as well as by the intersection of two planes. The array of coordinates {L,M,N;P,Q,R} is known as the *ray coordinates* for a line. The line can be considered as a *ray* of light joining two distinct points on the line. The array of coordinates {P,Q,R; L,M,N} is known as the *axis coordinates* for the line determined by the meet of two planes. In this case, the line can be considered as the *axis* of a pencil of planes. Pairs of dual elements, i.e., points or planes can thus form a line. It is because of this that a line in three-dimensional space is considered to be dual with itself or self-dual. A distinction is made in notation when the line coordinates are specified. The ray and axis coordinates are represented by $\hat{\mathbf{s}} = \{\mathbf{S};\mathbf{S}_0\}$ and $\hat{\mathbf{S}} = \{\mathbf{S}_0;\mathbf{S}\}$ respectively.

Statics

Consider a force with magnitude f acting on a rigid body as shown in Figure 2.3. The force is acting on a line **\$** with ray coordinates $\hat{\mathbf{s}} = \{\mathbf{S};\mathbf{S}_0\}$ where $|\mathbf{S}|=1$. The force f can be expressed as a scalar multiple $f\mathbf{S}$ of the unit vector **S** which is bound to the line **\$**. In order to add forces it is necessary to introduce a reference point O. The moment of the force **f** about this reference point, i.e., $\mathbf{m}_0$ can be expressed as $\mathbf{m}_0=\mathbf{r}\times\mathbf{f}$ where **r** is a vector to any point on the line **\$**. This moment may also be expressed as a scalar multiple $f\mathbf{S}_0$ where $\mathbf{S}_0$ is the moment vector of the line **\$**, i.e., $\mathbf{S}_0=\mathbf{r}\times\mathbf{S}$. The action of the force on the body thus can be expressed as a scalar multiple of the unit line vector, and the coordinates for the force are given by

$$\hat{\mathbf{w}} = f\hat{\mathbf{s}} = f\{\mathbf{S};\mathbf{S}_0\} \tag{2.19}$$

where $\mathbf{S}\cdot\mathbf{S}=1$ and $\mathbf{S}\cdot\mathbf{S}_0=0$. The above equation can be expressed in the form

$$\hat{\mathbf{w}} = f\hat{\mathbf{s}} = \{\mathbf{f};\mathbf{m}_0\}\,. \tag{2.20}$$

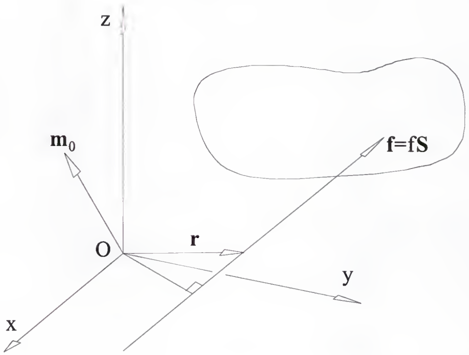


Figure 2.3: Force acting on a rigid body.

It can be seen from Figure 2.3 and $\mathbf{m}_0=\mathbf{r}\times\mathbf{f}$ that the vector $\mathbf{m}_0$ changes as the location of the reference point is changed. When the line **$** passes through the reference point, $\mathbf{m}_0=0$ and the coordinates of the force are $\{\mathbf{f};0\}$. Clearly, **f** is a line bound vector which is invariant with a change of coordinate systems while $\mathbf{m}_0$ is origin dependent.

Consider a rigid body on which is acting a force $\{\mathbf{f}_1;\mathbf{m}_{01}\}$ whose coordinates are expressed in terms of the reference point O. Equal and opposite collinear forces whose coordinates are $\{\mathbf{f}_1;\mathbf{m}_{02}\}$ and $-\{\mathbf{f}_1;\mathbf{m}_{02}\}$ are applied to the body so that the net resultant force and couple acting on the body remain the same. The forces $\{\mathbf{f}_1;\mathbf{m}_{01}\}$ and $-\{\mathbf{f}_1;\mathbf{m}_{02}\}$

may be replaced by a couple which is equal to the sum of the moments of the two line-bound forces about point O, i.e. $\mathbf{m}=\mathbf{m}_{01}-\mathbf{m}_{02}$. The combination of the force $\{\mathbf{f}_1;\mathbf{m}_{02}\}$ and the couple $\{\mathbf{0};\mathbf{m}\}$ has the same effect on the rigid body as the original force $\{\mathbf{f};\mathbf{m}_{01}\}$. Now consider an arbitrary system of forces with coordinates $\{\mathbf{f}_1;\mathbf{m}_{01}\}$, $\{\mathbf{f}_2;\mathbf{m}_{02}\}$, ..., $\{\mathbf{f}_n;\mathbf{m}_{0n}\}$ acting on a rigid body. It is assumed at the outset that a reference point O has been chosen, the magnitudes of the forces f_1, f_2, ..., f_n are specified, and the unit coordinates of the lines of action of the forces $\{\mathbf{S}_1;\mathbf{S}_{01}\}$, $\{\mathbf{S}_2;\mathbf{S}_{02}\}$, ...$\{\mathbf{S}_n;\mathbf{S}_{0n}\}$ are known. The forces are translated to point O and moments $\mathbf{m}_{01}$, $\mathbf{m}_{02}$, ... $\mathbf{m}_{0n}$ are introduced to yield an equivalent system of forces and torques that act on the rigid body. The net force acting on the rigid body is given by

$$\mathbf{f} \;=\; \sum_{i=1}^{n} \mathbf{f}_i \,. \tag{2.21}$$

and the line of action of this force passes through point O. The moment acting on the rigid body is given by

$$\mathbf{m} \;=\; \sum_{i=1}^{n} \mathbf{m}_{0i} \,. \tag{2.22}$$

The net force $\mathbf{f}$ (which passes through point O and whose coordinates are written as $\{\mathbf{f};\mathbf{0}\}$) and moment $\mathbf{m}_0$ (which is a non-line-bound vector whose coordinates are written as $\{\mathbf{0};\mathbf{m}_0\}$) are equivalent to the original set of forces. The coordinates of the resultant of the force and moment may be written as the sum of $\{\mathbf{f};\mathbf{0}\}$ and $\{\mathbf{0};\mathbf{m}_0\}$ as

$$\hat{\mathrm{w}} = \{\mathbf{f};\mathbf{m}_0\} \tag{2.23}$$

In general $\mathbf{f}$ and $\mathbf{m}_0$ will not be perpendicular and the quantity with coordinates $\hat{\mathrm{w}}=\{\mathbf{f};\mathbf{m}_0\}$, where $\mathbf{f}\cdot\mathbf{m}_0\neq 0$, was defined as a dyname by Plücker. It follows that in general it is not possible to translate the line of action of $\mathbf{f}$ through some point other than point O and have the translated force produce the same net effect on the rigid body as the original dyname. The moment $\mathbf{m}_0$, however, may be resolved into two components $\mathbf{m}_a$ and $\mathbf{m}_t$ as

$$\mathbf{m}_0 = \mathbf{m}_a + \mathbf{m}_t \tag{2.24}$$

which are parallel and perpendicular to the force **f** respectively. The moment $\mathbf{m}_a$ may be determined as

$$\mathbf{m}_a = (\mathbf{m}_0 \cdot \mathbf{S})\,\mathbf{S} \tag{2.25}$$

where **S** is a unit vector in the direction of the resultant force **f**. The moment $\mathbf{m}_t$ is then determined as

$$\mathbf{m}_t = \mathbf{m}_0 - \mathbf{m}_a \tag{2.26}$$

The line of action of force **f** may now be translated so that the force with coordinates $\{\mathbf{f};\mathbf{m}_t\}$ plus the moment $\{\mathbf{0};\mathbf{m}_a\}$ is equivalent to the dyname $\{\mathbf{f};\mathbf{0}\}+\{\mathbf{0};\mathbf{m}_0\}$(Figure 2.4). The dyname can thus be represented uniquely by a force **f** acting on the line of action $\{\mathbf{S};\mathbf{S}_{0t}\}$ where $\mathbf{S}_{0t} = \frac{\mathbf{m}_{0t}}{\mathrm{f}}$ and a parallel couple $\mathbf{m}_a$. *This representation is called a wrench and is attributed to Ball.*

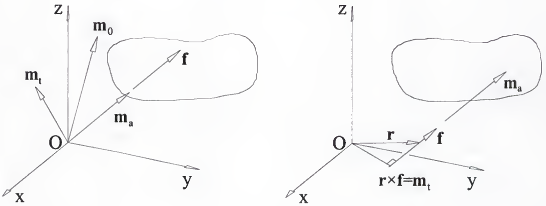


Figure 2.4: A Dyname and a Wrench.

The wrench $\hat{\mathbf{w}}$ which is equivalent to the dyname $\{\mathbf{f};\mathbf{m}_0\}$ may be written as

$$\hat{\mathbf{w}} = \{\mathbf{f};\mathbf{m}_0\} = \{\mathbf{f};\mathbf{m}_t\} + \{\mathbf{0};\mathbf{m}_a\} \tag{2.27}$$

It is desirable to express the wrench in terms of f and $\mathbf{m}_0$. Since $\mathbf{m}_a$ is parallel to f

$$\mathbf{m}_a = h\mathbf{f} \tag{2.28}$$

where h is a nonzero scalar which is called the *pitch* of the wrench. By algebraic manipulation and using Equation (2.24), the pitch of the wrench is given by

$$h = \frac{\mathbf{f}\cdot\mathbf{m}_0}{\mathbf{f}\cdot\mathbf{f}} \tag{2.29}$$

Clearly the pitch h is an invariant quantity. From (2.26) and (2.27) we have

$$\hat{\mathbf{w}} = \{\mathbf{f};\mathbf{m}_0 - \mathbf{m}_a\} + \{\mathbf{0};\mathbf{m}_a\} \tag{2.30}$$

and substituting $\mathbf{m}_a$ from (2.28) gives

$$\hat{\mathbf{w}} = \{\mathbf{f};\mathbf{m}_0 - h\mathbf{f}\} + \{\mathbf{0};\mathbf{m}_a\}. \tag{2.31}$$

The homogeneous coordinates for the line of action of the wrench are $\{\mathbf{f};\mathbf{m}_0\text{-}h\mathbf{f}\}$ and the equation for the line is therefore

$$\mathbf{r}\times\mathbf{f} = \{\mathbf{f};\mathbf{m}_0 - h\mathbf{f}\}. \tag{2.32}$$

In the same way as the action of a force upon a body can be expressed as a scalar multiple of a unit line vector, a wrench acting on a body can be elegantly expressed as a scalar multiple of a unit screw where

$$\hat{\mathbf{s}} = \{\mathbf{S};\mathbf{S}_0\} \tag{2.33}$$

and $\mathbf{S}\cdot\mathbf{S} = 1$. The pitch of the screw is given by

$$h = \mathbf{S}\cdot\mathbf{S}_0. \tag{2.34}$$

From (2.33)

$$\hat{\mathbf{s}} = \{\mathbf{S};\mathbf{S}_0\} = \underbrace{\{\mathbf{S};\mathbf{S}_0 - h\mathbf{S}\}}_{\textit{line}} + \underbrace{\{\mathbf{0};h\mathbf{S}\}}_{\textit{couple}} \tag{2.35}$$

The Plücker coordinates of the screw axis are $\{\mathbf{S};\mathbf{S}_0 - h\mathbf{S}\}$ and the equation of the axis is

$$\mathbf{r} \times \mathbf{S} = \mathbf{S}_0 - h\mathbf{S}. \tag{2.36}$$

A screw can therefore be regarded as a line with a pitch.

Kinematics

In this section a brief overview of the results relevant to the approaches in this research are presented. References are provided appropriately for the reader interested in details about the derivation and proofs of the results. The results presented here pertain to the velocity and acceleration analyses of serial manipulators (open chains). The detailed description of these analyses is presented by Rico and Duffy [Mar96, Ric99].

The *velocity analysis* of a serial chain consists of two parts. The first part relates the angular velocity of two arbitrary links in the serial chain, while the second part relates the velocity of a point fixed in one link as observed from different links (or reference frames) of the serial chain. The angular velocity of a rigid body *m* with respect to a body (or reference frame) *j*, can be expressed, in terms of auxiliary bodies (or reference frames), *m*-1, *m*-2, . . . , *j*+2 and *j*+1 as follows:

$${}^{j}\boldsymbol{\omega}^{m} \;=\; {}^{j}\boldsymbol{\omega}^{j+1} + {}^{j+1}\boldsymbol{\omega}^{j+2} + \ldots + {}^{m-2}\boldsymbol{\omega}^{m-1} + {}^{m-1}\boldsymbol{\omega}^{m}. \tag{2.37}$$

where ${}^{n-1}\boldsymbol{\omega}^{n}$ represents the magnitude and the direction of the angular velocity of body n as measured with respect to body n-1. It should be noted that in this proposition, it is not necessary that the bodies be physically connected at all.

Now consider a point O fixed in body *m* of a serial chain with respect to the reference frame *j*. The velocity of point O in body *m* as measured with respect to reference frame *j* is given by

$$ {}^{j}\mathbf{v}_O^m = {}_{j}\omega_{j+1}\ {}^{j}\mathbf{S}_0^{j+1} + {}_{j+1}\omega_{j+2}\ {}^{j+1}\mathbf{S}_0^{j+2} + \ldots + {}_{m-2}\omega_{m-1}\ {}^{m-2}\mathbf{S}_0^{m-1} + {}_{m-1}\omega_m\ {}^{m-1}\mathbf{S}_0^m \quad (2.38) $$

where ${}_{n-1}\omega_n$ represents the scalar magnitude of the angular velocity of body n as measured with respect to body n-1.

The *velocity state* of a rigid body *m* with respect to a reference frame *j*, is defined as

$$ {}^{j}\mathbf{V}^m = \begin{bmatrix} {}^{j}\boldsymbol{\omega}^m \\ {}^{j}\mathbf{v}_0^m \end{bmatrix} \quad (2.39) $$

The velocity state parameters ${}^{j}\boldsymbol{\omega}^m$ and ${}^{j}\mathbf{v}_0^m$ are often referred to as the screw coordinates, **\$**, and are written as

$$ {}^{j}\$^m = \{\ {}^{j}\boldsymbol{\omega}^m\ ;\ {}^{j}\mathbf{v}_0^m\ \}. \quad (2.40) $$

As an example, consider a two-link manipulator. The velocity state of link 2 with respect to the base link 0 can be expressed as

$$ {}^{0}\$^2 = {}^{0}\$^1 + {}^{1}\$^2, \quad (2.41) $$

where

$$ {}^{0}\$^2 = \begin{bmatrix} {}^{0}\boldsymbol{\omega}^2 \\ {}^{0}\mathbf{v}_0^2 \end{bmatrix} = {}_{0}\omega_2 \begin{bmatrix} {}^{0}\mathbf{S}^2 \\ {}^{0}\mathbf{S}_0^2 \end{bmatrix}, \quad (2.42) $$

$$ {}^{0}\$^1 = \begin{bmatrix} {}^{0}\boldsymbol{\omega}^1 \\ {}^{0}\mathbf{v}_0^1 \end{bmatrix} = {}_{0}\omega_2 \begin{bmatrix} {}^{0}\mathbf{S}^1 \\ {}^{0}\mathbf{S}_0^1 \end{bmatrix}, \quad (2.43) $$

$$ {}^{1}\$^2 = \begin{bmatrix} {}^{1}\boldsymbol{\omega}^2 \\ {}^{1}\mathbf{v}_0^2 \end{bmatrix} = {}_{1}\omega_2 \begin{bmatrix} {}^{1}\mathbf{S}^2 \\ {}^{1}\mathbf{S}_0^2 \end{bmatrix}, \quad (2.44) $$

and ${}^{i}\mathbf{S}^{j}$ are unit vectors.

In the *forward velocity analysis*, the angular velocity for each revolute and helical joint and the linear velocity for each prismatic joint are known. The objective is to determine the velocity state of the last link of the manipulator relative to a reference frame (base). The velocity state can simply be written as (e.g., 6-link manipulator)

$$\begin{bmatrix} {}^0\boldsymbol{\omega}^6 \\ {}^0\mathbf{v}_0^6 \end{bmatrix} = {}_0\omega_1 \begin{bmatrix} {}^0\mathbf{S}^1 \\ {}^0\mathbf{S}_0^1 \end{bmatrix} + {}_1\omega_2 \begin{bmatrix} {}^1\mathbf{S}^2 \\ {}^1\mathbf{S}_0^2 \end{bmatrix} + {}_2\omega_3 \begin{bmatrix} {}^2\mathbf{S}^3 \\ {}^2\mathbf{S}_0^3 \end{bmatrix} + {}_3\omega_4 \begin{bmatrix} {}^3\mathbf{S}^4 \\ {}^3\mathbf{S}_0^4 \end{bmatrix} + {}_4\omega_5 \begin{bmatrix} {}^4\mathbf{S}^5 \\ {}^4\mathbf{S}_0^5 \end{bmatrix} + {}_5\omega_6 \begin{bmatrix} {}^5\mathbf{S}^6 \\ {}^5\mathbf{S}_0^6 \end{bmatrix} \tag{2.45}$$

This may be written as

$$\begin{bmatrix} {}^0\boldsymbol{\omega}^6 \\ {}^0\mathbf{v}_0^6 \end{bmatrix} = {}_0\omega_1\,{}^0\$^1 + {}_1\omega_2\,{}^1\$^2 + {}_2\omega_3\,{}^2\$^3 + {}_3\omega_4\,{}^3\$^4 + {}_4\omega_5\,{}^4\$^5 + {}_5\omega_6\,{}^5\$^6 \tag{2.46}$$

where ${}^i\$^j$ are the appropriate unitized screw coordinates for each joint axis. This may be rewritten in the matrix format

$$\begin{bmatrix} {}^0\boldsymbol{\omega}^6 \\ {}^0\mathbf{v}_0^6 \end{bmatrix} = {}_0\omega_6\,{}^0\$^6 = \mathbf{J}\,\boldsymbol{\omega} \tag{2.47}$$

where **J**, called the *Jacobian* matrix is the 6x6 matrix formed from the unitized screw coordinates as

$$\mathbf{J} = \begin{bmatrix} {}^0\$^1 & {}^1\$^2 & {}^2\$^3 & {}^3\$^4 & {}^4\$^5 & {}^5\$^6 \end{bmatrix} \tag{2.48}$$

and $\boldsymbol{\omega}$ is the vector of joint velocities given as

$$\boldsymbol{\omega} = \begin{bmatrix} {}_0\omega_1 & {}_1\omega_2 & {}_2\omega_3 & {}_3\omega_4 & {}_4\omega_5 & {}_5\omega_6 \end{bmatrix}^T . \tag{2.49}$$

From the position analysis all the elements of **J** are known, coupled with the angular velocities being known, (2.47) can be solved to determine the velocity state of the end effector. For the *reverse velocity analysis*, the Jacobian matrix is again known coupled with the velocity state of the end-effector relative to the base. The individual joint velocities can be obtained from

$$\boldsymbol{\omega} = \mathbf{J}^{-1}\begin{bmatrix} {}^{0}\boldsymbol{\omega}^{6} \\ {}^{0}\mathbf{v}_{0}^{6} \end{bmatrix}. \tag{2.50}$$

If the Jacobian matrix is singular, and thus its inverse cannot be obtained, the manipulator is referred to as being in a singular configuration.

For the acceleration analysis, open serial chain is considered, where body m is an arbitrary link and the rigid body j is the reference frame. Further, it is assumed that adjacent bodies or links of the serial chain are connected by helical pairs, and O is an arbitrary point fixed in the reference frame. The reduced acceleration state of an arbitrary link m, with respect to the reference frame j, is defined by

$${}^{j}\mathbf{A}_{R}^{m} = \begin{bmatrix} {}^{j}\boldsymbol{\alpha}^{m} \\ {}^{j}\mathbf{a}_{0}^{m} - {}^{j}\boldsymbol{\omega}_{0}^{m} \times {}^{j}\mathbf{v}_{0}^{m} \end{bmatrix}, \tag{2.51}$$

and in terms of screws connecting adjacent links, is given as

$$\begin{aligned} {}^{j}\mathbf{A}_{R}^{m} = {} & {}_{j}\dot{\omega}_{j+1}\ {}^{j}\$^{j+1} + {}_{j+1}\dot{\omega}_{j+2}\ {}^{j+1}\$^{j+2} + \quad \ldots \quad + {}_{m-2}\dot{\omega}_{m-1}\ {}^{m-2}\$^{m-1} + {}_{m-1}\dot{\omega}_{m}\ {}^{m-1}\$^{m} \\ & + [{}_{j}\omega_{j+1}\ {}^{j}\$^{j+1} \quad {}_{j+1}\omega_{j+2}\ {}^{j+1}\$^{j+2}] \\ & + [{}_{j}\omega_{j+1}\ {}^{j}\$^{j+1} + {}_{j+1}\omega_{j+2}\ {}^{j+1}\$^{j+2} \quad {}_{j+1}\omega_{j+2}\ {}^{j+1}\$^{j+1}] \\ & + \ldots \\ & + [{}_{j}\omega_{j+1}\ {}^{j}\$^{j+1} \quad {}_{j+1}\omega_{j+2}\ {}^{j+1}\$^{j+2} + \ldots + {}_{m-3}\omega_{m-2}\ {}^{m-3}\$^{m-2} \quad {}_{m-1}\omega_{m}\ {}^{m-2}\$^{m-1}] \\ & + [{}_{j}\omega_{j+1}\ {}^{j}\$^{j+1} + \ldots + {}_{m-3}\omega_{m-2}\ {}^{m-3}\$^{m-2} + {}_{m-1}\omega_{m}\ {}^{m-2}\$^{m-1} \quad {}_{m-1}\omega_{m}\ {}^{m-1}\$^{m}] \end{aligned} \tag{2.52}$$

where $[{}^{j}\$^{j+1} \quad {}^{k}\$^{k+1}]$ is the Lie product of the screws inside the bracket. This can also be written as

$${}^{j}\mathbf{A}_{R}^{m} = [\mathbf{J}]\dot{\boldsymbol{\omega}} + [\mathbf{L}], \tag{2.53}$$

where **J** is the Jacobian and **L** is the summation of all the Lie products.

The Lie product of two screws is also known as the Dual Vector Product and is defined in the following manner. Consider two screws $\$_1$ and $\$_2$,

$$(\$_1 \quad \$_2) = \left(\left(\vec{\omega}_1 \,;\, \vec{\upsilon}_{01}\right) \quad \left(\vec{\omega}_2 \,;\, \vec{\upsilon}_{02}\right)\right) \tag{2.54}$$

then the Lie product is

$$[\$_1 \quad \$_2] = \left[\left(\vec{\omega}_1 \times \vec{\omega}_2\right) \;;\; \left(\vec{\omega}_1 \times \vec{\upsilon}_{02} - \vec{\omega}_2 \times \vec{\upsilon}_{01}\right)\right]. \tag{2.55}$$

Rico and Duffy have further extended this analysis to the jerk [Ric99].

CHAPTER 3
REVIEW OF LITERATURE

Conventionally, path tracking falls under the umbrella of robot control algorithms, which are typically divided into two stages: path or trajectory planning and path tracking (path control). Path tracking is generally considered the lower level of control and path or trajectory planning is considered the higher level. The path tracker attempts to make the robot's actual position and velocity match some desired values of position and velocity, which are provided to the controller by the path planner. Although the terms *path* and *trajectory* are used interchangeably, a distinction is sometimes made: the *path* is a time-independent description of the route tracked by the manipulator, and the *trajectory* is a time-dependent description of the route.

A variety of approaches have been undertaken to improve the path following performance of robots. These approaches vary from modifying the 'path planner' to changing the 'path tracker' to an amalgam of both. In some cases, the distinction between the two is transparent or is lost. Other approaches attempt to optimize a particular criterion in robot control, for example, tracking time or joint torque. The approach taken in this research is different because it uses Screw Theory for path tracking. Literature on this research is lacking. Nonetheless, I present an overview of the techniques that are most relevant in the following sections. I attempt to group literature under broad categories although the reader is warned that each approach is relatively distinct.

Techniques to Improve/Modify the Path Planner

Robot trajectory refers to a time history of position, velocity, and acceleration for each degree of freedom. Many robot applications require the end-effector to track a specified path for the purpose of accomplishing the task satisfactorily and avoiding obstacles. For achieving this objective, the trajectory planning can be conducted either in the joint-variable space or in the Cartesian space.

The earliest work of Shin and McKay [Shi85] attempts to improve the trajectory planning stage of robotic manipulator control. The authors claim that robot control is divided into path planner and path tracker components because the process of robot control, if considered in its entirety, is complicated. Dividing the controller into two parts makes the process simpler. The path tracker is generally a linear controller and the non-linearities of manipulator dynamics frequently are not taken into account at this level. The simplicity obtained from this division comes at the expense of efficiency and the source of this inefficiency is the trajectory planner. The authors assert that to be able to use the robot at maximum efficiency the trajectory planner must be aware of the robot's dynamic properties. To partially alleviate the inefficiency of the planner the authors present a solution to the minimum-time manipulator control problem[1] subject to constraints on its geometric path and input torques/forces. The output of the planner is the true minimum-time solution.

The problem is set up by defining a dynamic model for the manipulator using the Lagrangian formulation. The motion is constrained to a fixed path in joint space and the path is given as a parameterized curve $f(\lambda)$, where λ varies from 0 to λ_{max}. The derivative of λ with respect to time is taken as μ. The resulting equation is in terms of λ and μ. If the parameter λ is considered as arc length in Cartesian space, then μ is the speed and $\dot{\mu}$ the acceleration along the geometric path. After setting up the dynamic equations, the control problem is addressed. From optimization, the minimum cost is equated with minimum time, which results in optimizing the operating speed of the robot. This is called Minimum Time Path Planning (MTPP).The cost function is optimized with the constraints on the velocities, slope of trajectories, and joint torques, as equalities or inequalities, thus yielding regions of admissibility and islands of inadmissibility. These regions provide the acceptable bounds for trajectories. Then an algorithm called the Algorithm for Constructing Optimal Trajectories (ACOT) is used to determine the optimal trajectories. The optimal trajectories are determined by using the algorithm with the bounds determined previously using MTPP. The optimal trajectory may actually touch the boundary of the admissible region, generating a dangerous case. It is pointed out however, that if slightly conservative torque bounds are used in the calculations, then the actual admissible region will be slightly larger than the calculated admissible region, giving some margin of error.

Previously, Luh et al. [Luh77, Luh81] had proposed the use of linear and nonlinear programming in trajectory planners to generate desired positions, velocities, and accelerations. These methods assumed that the desired path was given in terms of the path's endpoints and set of intermediate points or corner points. Along each segment of the path the (constant) maximum accelerations and velocities were given. In general, these constant bounds may have been quite inaccurate for some parts of the segment, since worst-case bounds for the whole segment had to be used. Shin et al.'s [Shi85]

[1]The problem of moving a manipulator in minimum time along a specified geometric path.

method was an improvement over these techniques since it calculated the acceleration and velocity limits directly from the path, the manipulators dynamic equations, and actuator characteristics.

Another approach was taken by Shin and McKay [Shi86a] in an attempt to address some issues that were not fully addressed by the MTPP method [Shi85]. The authors point out that in the MTPP approach, the solution works only for minimum-time problems. In situations where driving the robot consumes large amounts of power, the assumption that minimum time is equivalent to minimum cost may not be valid. Another assumption is that joint torques can be changed instantaneously. Also, sometimes the actuator torque limits are dependent on one another as in the case when a robot uses a common power supply for the servoamplifiers for all joints. To rectify the shortcomings of the previous approach, the idea of using Dynamic Programming was proposed to find the optimal phase plane trajectory (λ vs. μ). The advantage of using Dynamic Programming over previous methods that used linear or nonlinear programming [Luh77, Luh81] is that it places few restrictions on the cost function that is to be minimized. Putting limits on jerk is possible, and the interdependence of torque bounds can be handled easily.

The authors call this approach the minimum-cost trajectory planning (MCTP). The obstacle avoidance problem is bypassed as it is assumed that the path has been specified *a priori* and the problem becomes: what controls will drive the robot along a specified curve in joint space with minimum cost, given constraints on velocities, control signals and their derivatives? As done in their previous work, the authors reduce the complexity of the control problem by introducing a single parameter that describes the robot's position in the form of a parameterized curve in the robots joint space.[2] This single parameter and its time derivative completely describe the current state of the robot. The problem then becomes a two-dimensional minimum-cost control problem with some state and input constraints.

To apply dynamic programming to this problem, in which the dimensionality has been reduced to two state variables, the phase plane is divided into a grid. The cost of going from one point in the grid to next point is calculated. Since all the dynamic equations are strictly given in terms of the state variables, the cost computation is done entirely in phase coordinates. Once costs have been computed, the dynamic programming algorithm is applied to determine positions, velocities, and torques, from the resulting optimal trajectory, and the curve and dynamic equations.

The significance of the results is that they provide a simple approximation to the truly optimal trajectory problem to any desired degree of accuracy. It must be noted however that the effect of grid size on the quality of results is of practical importance. The grid should be fine enough to provide good results but not too fine to cause excessive computations. The varying of column size and row size in the grid will yield different effects on the results. Varying the number of columns (λ divisions) varies the accuracy of the dynamic model while varying the number of rows (μ divisions) varies the accuracy of the approximation to the true minimum-cost solution. Another important factor is the ratio of the number of rows to the number of columns so that the slope of the curve connecting two adjacent points in the grid is small.

[2] The task planner generates a sequence of Cartesian points and additional intermediate knot points which, if necessary, are transformed pointwise to points in joint space. These joint points are interpolated to form a geometric path in joint space [Pau81].

Another paper authored by Shin and McKay [Shi86b] addresses the issue of finding the geometric path. The problem states that given the solution to the minimum time trajectory planning problem, find the geometric path or the trajectory curve so as to minimize the traversal time.

Xiangrong and Xiangfeng [Xia94] present a method for trajectory planning that includes Cubic Polynomial (CP) motion trajectory planning and addresses the minimum-time trajectory problem. Their work extends and improves the work done on a similar problem by Luh et al. [Lin83]. It uses Cartesian space schemes for trajectory planning. The characteristics of CP motion are that the end-effector not only must achieve the final target point, but also must move along the desired path. The time history of the positions, velocities, and accelerations of the end-effector are assumed to be prescribed, and the corresponding joint displacements, velocities, and accelerations are obtained from a given position and orientation of the end-effector by solving the inverse kinematics.

If T is the time needed for a robot to perform a CP motion, the time interval 0-T is divided into smaller sub-intervals. At each interval computations are done for the transformation matrix for the end-effector coordinate with respect to the base coordinate using the equations of the path and constraint conditions for the end-effector position and orientation. Inverse kinematics is used to determine the velocities and accelerations of each joint. After this information is obtained for two successive intervals, a 3rd to 5th order polynomial is used to specify each joint trajectory in that sub-interval. If there are no constraints on velocities of each joint except at initial and final points, a cubic spline fit can be used for all intermediate points and a 4th order for the first and last point. The coefficients of polynomials at each segment can be obtained recursively and hence the trajectories for all sub-intervals are planned.

The next step is the Minimum-Time Trajectory planning. It is assumed that the maximum velocity, acceleration, jerk, and torque for all joints are specified and are used as constraints. A new scaling factor is determined which attempts to maximize (≈ 1) the ratio of current value/maximum value (of velocity, acceleration, etc) during a sub interval. A new trajectory is determined recursively using the former method until the maximum ratio is obtained at which point the minimum-time joint trajectories are considered obtained.

The paper discusses an additional issue of determining the number of time sub-intervals. A maximum allowable deviation of trajectory is assumed. Using the CP Trajectory algorithm the trajectory is planned. From this information the corresponding actual position of the end-effector in Cartesian space is determined. This is compared with the trajectory, and if it is not within the allowable deviation, the sub-interval is divided into two parts by adding an intermediate point and the same process is repeated until an acceptable time interval is obtained.

Piazzi and Visioli [Pia97] have presented an approach to find the minimum-jerk cubic spline joint trajectory of a robot manipulator using interval analysis. Minimum-jerk trajectories are desirable for their amenability to path tracking and to limit robot vibrations while cubic splines are used to ensure continuity of velocities and accelerations in the robot movement.

The importance of minimizing the jerk in trajectory planning is primarily due to the fact that joint position errors increase when the jerk increases. Moreover, minimum-jerk joint trajectories are desirable for their similarity to human joint movements and to limit robot vibrations. For achieving this goal, a global optimization method, based on interval analysis, is presented.

The path of the manipulator is specified in Cartesian coordinates and is converted to joint space via inverse kinematics. The sequences of joint displacements are interpolated by use of cubic polynomials to ensure an overall continuity of position, velocity, and acceleration. Using the total elapsed time for the displacement, a parameterized spline is obtained that incorporates continuity of positions and velocities. The optimal trajectory planning problem with minimum-jerk criterion is then posed as a constrained minimax optimization problem. The Interval Algorithm [Moo79] is used for the global optimization problem and is called the Minimum-Jerk Interval (MJI) algorithm. An example is presented for a 2DOF robot to illustrate the minimum-jerk joint-space trajectory planning with continuity of velocities and accelerations using the interval algorithm.

According to the robot's dynamic model that is employed, trajectory planning can be done either continuously or discretely. Techniques such as of [Shi85] employ a phase-plane technique to obtain the optimal solution for the trajectory planning problem. In these algorithms, constraints are considered to make full use of the capacities of the joint actuators and a true minimum time solution is yielded. It should be pointed out that these algorithms work for a small range of trajectory planning problems [Du95] that consider only generalized force constraints. Other realistic constraints such as joint velocity limits, jerk constraints, and end-effector velocity limit are neglected.

A discrete model is, however, more appropriate than a continuous model in solving the trajectory planning problem [Fla88]. In this method the whole generated path is discretized by individual points. The obtained trajectory of the discrete points that optimizes the given performance index is thus an optimal one. The trajectory planner presented by Flash [Fla88] is based on a linear programming model. The use of an objective akin to the minimum time objective rather than the true minimum time objective function makes it possible for linear programming to be used, which allows fast computation. However, in this method, robot dynamics are not considered. Tan and Potts [Tan88] extend this method by considering non-linear constraints, such as generalized forces and joint jerks, and use nonlinear programming to obtain the optimal solution. In their method, the discrete trajectory planner is modified by taking all time intervals to be the same, which is difficult to execute as it is computationally extensive.

A new method for segmental discrete trajectory planning, in which the whole trajectory is planned segment by segment along the path is presented by Du et al. [Du95]. The consistency of the time interval at the trajectory planning stage with the sampling time interval determined at the trajectory tracking control stage is considered. The method reduces the computation memory and run time.

Path planning is often times categorized as either being *static* or *dynamic*, depending on the mode of available information. Static path planning is used when all the information about obstacles is known *a priori*. Most path planning methods are static. Dynamic path planning is used when only partial a priori information is available about the obstacles, and the environment is unpredictable and time-varying. Zelek [Zel95] has presented an approach for dynamic path planning. Dynamic path planning is problematic because the path needs to be continually recomputed as new information becomes available. Dynamic path planning is accomplished by using a harmonic function as the potential function to model the free space. Computation of the harmonic function is formulated as an iterative process and is performed independently of the execution of trajectory commands. In order to achieve continuous robot navigation with guaranteed control, there are coarse-to-fine set of harmonic computations being continuously carried out in parallel.

Techniques to Improve Path Tracking

As alluded to before, since robot dynamics are highly nonlinear, coupled, and of high dimension, time-optimal control is difficult: to simplify the problem it is commonly divided into offline trajectory planning and online tracking. The papers reviewed above have addressed only the trajectory planning problem, and have done so considering only the robot dynamics and not the tracking controller dynamics. These solutions have used the path constraint to reduce the dimension of the problem to 2, the path position and velocity, and solve the problem using a variety of optimizing techniques.

These solutions provide mathematically correct time-optimal trajectories for any given robot model. However, their application to real systems is not so straightforward. Because the robot dynamics are never precisely known, the resulting trajectories are too approximate to be applied open-loop. The path constraint, which is so useful in reducing the dimension of the optimal control problem, preempts consideration of tracking error. The resulting time-optimal solutions push actuators to their limits, leaving no margin for controlling tracking errors.

Some published results have addressed these issues, including an approach by Shin et al. [Shi87] that proposes planning trajectories that are robust to payload uncertainties. Cahill, Kieffer, and James [Cah96] proposed a scheme for planning robust "time-optimal" trajectories for robots under computed-torque control. This scheme was based on the experimental identification of un-modeled dynamics as joint acceleration disturbances and a theory for relating those disturbances to torque margins that should be held on reserve during trajectory planning. The scheme also involved the selection of controller gains to ensure that tracking is accurate to a prescribed tolerance.

Kieffer et al. [Kie96] in continuation of their previous work [Cah96] extend the approach to a more general architecture for online trajectory generation. Their work proposes a state feedback law that ensures nearly time-optimal path tracking which is accurate to a prescribed tolerance in the presence of experimentally identified levels of disturbance. The input path acceleration ($\ddot{s}$) is generated online as a feedback function of the plant and trajectory generator states and is thus called a *closed-loop trajectory generator*.

The model for the robot is developed which is inclusive of joint acceleration disturbances. These disturbances can be identified from online measurements. The tracking error is then predicted and compensation torques are computed based on the error. A methodology is developed for choosing controller gains to meet a prescribed tolerance for tracking accuracy and for predicting levels of compensation torque that will develop online. With the theory developed to this point, the authors had previously employed dynamic programming to robustly determine time-optimal trajectories [Cah96].

In the current instance, they extended it to Robust *Closed-Loop Trajectory Generation*. The idea is to determine how much the path acceleration ($\ddot{s}$) can be modified without saturating any actuators which is necessitated by the fact that the equations predicting the tracking error and compensation torque are only valid if the torque commands are implemented without clipping.

The closed loop architecture was implemented on the first two links of a SCARA robot arm. The robot model, identified by least squares techniques, included all rigid body inertial effects as well as independent coulomb and viscous friction terms for each joint. The task was chosen to be the same as the one used when the previous technique was implemented for comparison purposes. The results were successful, as the path velocity of the closed-loop scheme was everywhere greater than or equal to that of the previous scheme. The total elapsed time for traversing the trajectory was also improved and was very close to the model's time-optimal elapsed time. Torque utilization also increased substantially and exact saturation occurred most of the time. Also, the closed-loop system tracked the path within the specified tolerance, and that the errors were very similar to those of the previous robust trajectory planning scheme. It was concluded that the tracking times were reduced while the tracking accuracy and robustness remained approximately the same. Keiffer et al. have presented both the techniques with extensive experimental results in [Kie97].

Another approach to address the tracking problem involves developing methods for path planning *and* path tracking simultaneously. This approach is taken by Tsuchiya and Egami [Tsu90], and they assert that the best whole system is synthesized by combining them suitably. They have proposed robot path control with variable speed trajectory planning and an assortment of control methods for tracking. They present a new trajectory planning method that takes the control ability of the designed manipulator control system into consideration. This is done systematically without trial and error on the basis of the frequency response of the designed control system. This trajectory planning method is implemented with the following control methods: Adaptive control, Preview control, and Zero-error tracking control.

An important issue at the trajectory planning stage is how to utilize the freedom of tracking speed. The purpose of the trajectory planning method in this paper is how to carry out trajectory planning so that the system can utilize the maximum ability of the designed control system. The results show that the tracking speed of the robot manipulator in both the work space and joint space changes with the curvature of the given path. This trajectory planning method is known as "variable speed trajectory planning." The method proposed by the authors differs than the traditional variable speed trajectory planning in the sense that the ability of the designed control system is explicitly taken into account.

The methodology for the planning entails a series of procedures and starts by the synthesis of the manipulator control system. By using the Bode plot of the control system the maximum angular frequency that guarantees no gain deterioration and phase delay is determined. The minimum angular frequency (ω_0) amongst each joint servo system is determined. This is referred to as the standard angular frequency and the robot can track the path with an angular frequency lower than this. The given path in Cartesian space is then converted into joint coordinates. A reference point sequence is obtained by plotting equidistant points on the transferred path by assuming that the robot manipulator tracks the path with a constant speed. A circle is obtained consisting of three consecutive reference points on part of the tentative desired trajectory. From the circle, the angular frequency of the sinusoidal signal (ω_1) to be applied to the servo system to draw the circle is obtained. The points are taken as reasonable reference points on the path if $\omega_1 = \omega_0$, otherwise the points are moved in appropriate direction until the condition is satisfied. A circle consisting of three consecutive points approximates each part of the given path. By using this method a desired signal is obtained as a function of time that can be applied to each servomotor; this signal is always part of a sinusoidal function with fixed angular frequency.

Utilization of future information is useful for improvement of system response if available, and usually, the path tracked by a robot is available in advance. In this context, Preview Control utilizes future information and has many attractive characteristics over usual control systems. Although even simple application of preview action is effective, application along with variable speed trajectory planning is even more effective in reducing tracking error. In order to determine the desired signal for each servo system of each arm, trajectory planning is carried out based on the frequency response of preview control. A characteristic of this control system is the existence of feed-forward compensation utilizing a future desired signal. The transfer function of the preview control system from desired signal to controlled variable is obtained and the maximum angular frequency from the Bode plot is determined. The servo system can respond to the desired signal with angular frequency ω_0 without any gain deterioration or phase delay.

Another control method employed with variable speed trajectory planning is the Zero-error Tracking controller. Zero phase error tracking controller is proposed to cancel the effect of unstable zeroes of a stabilized closed-loop system. A unity gain controller can be derived on the basis of zero phase error controller when the angular frequency of the desired signal for the control system is known and fixed. In general, this type of controller is not useful because such a situation rarely exists in usual engineering problems. However, the condition of constant angular frequency of the desired signal is always satisfied by variable speed trajectory planning. Nonlinear compensation methods are further employed to eliminate the nonlinear effects.

The authors propose a third method for controlling the robot that employs the variable speed trajectory planning method. Adaptive Control is used as a countermeasure to parameter variations of the robot manipulator during an operation. Specifically, model reference adaptive control (MRACS) is applied to robot path control. By using the MRACS the control system is decoupled. This decoupling is very convenient for application of variable speed trajectory planning method as the method is essentially for single-input single-output systems. It requires plotting of Bode plot for each servo system.

Simulation results for the third method are presented. It is noted that the results for the other two control techniques were omitted, as the results were more or less the same. Adaptive control (MRACS) is used with constant speed and variable speed trajectory planning. It is noted however that robot path control is considerably improved by applying the proposed trajectory planning with the significance of the said method being its non-reliance on trial and error methods. The input torque restriction of the servomotors is not dealt with and reference is made to the work of Shin et al. [Shi85] that addresses this issue.

Abe and Tsuchiya [Abe92] published work that refined and detailed the work published earlier on variable-speed trajectory planning by Tsuchiya and Egami [Tsu90]. Wang, Tsuchiya, and Hshimoto [Wan93] presented two control methods for path tracking: a digital control method based on the principle of movement of a nonlinear mechanical system, and a preview control method utilizing future information of the desired trajectory to improve the tracking performance of the former. A digital control law was derived and disturbance terms (functions of position and velocity) were included that otherwise may have been unknown.

Wang and Tsuchiya [Wan94] proposed a new trajectory planning method based on high-level qualitative judgement. Fuzzy reasoning was used as the tool for this. The method is called fuzzy trajectory planning. This effort was to address unresolved problems in the previously proposed variable-speed trajectory planning method.

The proposed trajectory method takes into account the important efficiencies and limits of a robot manipulator. Fuzzy reasoning [Zad69] is considered a powerful tool for making brief judgements and with the algorithm being very simple, it is used for the trajectory planning problem at hand. The requirements for setting the trajectory planning are set as:

- The attained trajectory must be efficient for calculation and for putting into practice;
- Its position, velocity, and (except at the starting point) acceleration must be smooth functions of time; and
- Trajectory planning must be done without any trial and errors, keeping in consideration the limits of the actuator torques and the characteristics of the control system.

The variable speed trajectory planning suffered a few shortcomings in regards to the aforementioned requirements. The method required different processes for starting and stopping, i.e., the planning method had to be changed for the starting and ending part of the path. The method did not handle the torque limits on the actuators positively, and the algorithm was too complex. The fuzzy method proposed attempts to solve these issues at a certain level.

In fuzzy trajectory planning, the rough knowledge of the robot manipulator and the control system is first expressed at the language level. Then by using a membership function of the fuzzy set, these expressions are transformed into fixed quantities. And then trajectory planning is done using the fuzzy reasoning method. Therefore, the most important step in fuzzy trajectory planning is how to acquire the rough but important knowledge about the robot manipulator and the control system, and how to make them abstract.

Abe and Tsuchiya [Abe92] have already made use of the frequency characteristics of control systems and have explicitly taken the efficiencies of the control systems into account. The same information is adopted for fuzzy reasoning and for setting up a rule base. Next, the issue of velocity and torque limits is addressed. Once again, the variable speed trajectory planning provides the solution so that the velocity limits are not exceeded. The velocity of a robot manipulator moving along a path in a joint angle space can be determined by three items: the radius of curvature, the angular frequency of the trajectory, and the distance L from a sharp bend. All these are treated as fuzzy variables and constitute the fuzzy reasoning rules.

Once all the information is available at the language level, it is expressed in qualitative form by using membership functions of fuzzy sets. The rules for reasoning are set up. By using the rules, the reasoning results are obtained followed by defuzzification to get the final solution for the velocity to move along the path. While the issue of limiting torque is not addressed, it is pointed out that the information obtained from the reasoning results can be used to calculate the required torque. If this torque is within the prescribed limits then no action is needed. If not, then a search for a suitable new point near this point is done by moving along the path, taking very small steps. From the new point, the trajectory planning is carried out again by using fuzzy reasoning.

The authors concede that the problem of torque limits is a difficult problem to handle systematically at the trajectory planning stage and that although their method does consider torque limits, there may be cases where it exceeds the torque limit. The way to correct is to check for the exact value of torque, and if it exceeds the limit, reflect it immediately in trajectory planning. This will increase the processing time but the time of calculation for the fuzzy method is shorter than the variable-speed trajectory method.

In summary, this method can obtain the trajectory without trial and error by considering the efficiencies of the control system and motor limits. It has a fairly simple algorithm, which, in some cases, may allow trajectory planning in real time. Also, the position, velocity, and acceleration form a smooth function in time.

Grasa, Volberg, and Polyakhov [Gra93] present an approach that combines direct adaptive control and fuzzy logic control. To determine the coefficients and terms of the robot dynamic model and unmodeled dynamic effects, the authors use a combination of the computed torque method and the velocity gradient technique. This provides a simple way to obtain some known and unknown parameter adaptive control laws, in particular, the exponential path tracking adaptive control algorithms. The algorithms require online measurements of the tracking error and its first derivative. The fuzzy logic control strategy is employed to attain implementable algorithms based on the developed adaptive laws.

Many model-based controllers neglect non-linearities or system dynamics to simplify the design [Mas99]. These omissions may lead to poor performance or instability. This is especially true for Multi-Input Multi-Output (MIMO) systems. Mason, Walchko, and Crane [Mas99] have presented the development and implementation of MIMO hybrid fuzzy controller for a two-link robot manipulator to deal with the problem of designing MIMO control laws for systems with significant nonlinearities in the dynamics. The controller is aimed at providing better tracking performance of the robot manipulator. The method utilizes the designer's intuition (heuristic knowledge) and the mathematical description of the system. The controller combines the traditional proportional-derivative (PD) control law with a fuzzy controller in a MIMO framework. This is done because hybrid controllers provide a more defined control structure over fuzzy control while they improve the accuracy and portability of the controller.

The MIMO fuzzy controller is formulated by extending the SISO (single-input single-output) fuzzy concepts to the vector domain. The control vector is resolved into a direction, magnitude, and position-derivative ratio and by doing so the MIMO problem for solving for a control effort is reduced to a scalar one. The system is further simplified by making assumptions about the system (e.g., uncoupled), and the complexity of the fuzzy inference system is reduced.

For the purposes of benchmarking, a PD and a Feedback Linearization (FL) controller are also developed. Simulations are conducted to examine the performance of a fuzzy MIMO hybrid controller in ideal conditions. These results are compared against the PD and the FL schemes. Another set of simulations is done to study the effects of an unknown load disturbance. In the first case, the FL and the PD schemes produced a larger steady state error than the fuzzy scheme and their control effort was higher too. In the latter case, with random and deterministic disturbances, the fuzzy controller was better able to account for these disturbances. The control effort in this case was the same for all controllers. It was concluded that the MIMO fuzzy controller outperformed the other controllers. This scheme is more robust to random and deterministic disturbances and also offers portability to other robots with minimal effort.

Najson and Kreindler [Naj96] have proposed a discontinuous robust control law for rigid manipulators to track a desired trajectory. The proposed control laws exploit manipulator properties and do not require the measurement of accelerations and the inversion of the inertia matrix. They require the knowledge of the present value of the desired trajectory and its first and second derivatives, rather than an entire predetermined trajectory for a given task. This enables on-line task changes and model following can be accommodated.

The authors set out by deriving the dynamic equations of a rigid manipulator. The equations are transformed to a form suitable for adaptive control as established by Slotine and Li [Slo87]. A robust tracking control law is derived for the discontinuous case, which is a state-feedback control law. The global, exponential stability to zero tracking error of the law is proved by a Lyapunov method and has exponential stability. A continuous approximation of the control law is then developed. This is done to avoid chatter in the system; while this causes the error to not exponentially tend to zero, it causes it to ultimately converge to an arbitrarily small neighborhood. This is what is referred to as practical stability. Once the controller is designed, adaptive ideas are incorporated. The gains are determined adaptively instead of being fixed in advance.

An example is presented with simulation results for a two-link manipulator with parameter and disturbance uncertainties, including motor dynamics and friction. The continuous control law showed excellent robust tracking. The tracking error was reduced to near zero in about 0.8 seconds. It is noted that the control law does not take into account the uncertain motor dynamics added in the simulation.

Dakev, Chipperfield, and Fleming [Dak96] have presented an approach to determine approximate solutions of path following optimal control problems by exploiting modern global search and optimization techniques. According to the methodology developed, controls are represented by discrete vectors and substituted in the system equations. Alternatively, when it is possible to exclude the controls from the system equations, a discrete vector represents the parameterization function of the path. In either case, the components of these vectors are regarded as variables of a performance index based goal function that is to be minimized with respect to the control and phase constraints.

Employing such schemes allows the modeling and solution of both open-loop and closed-loop path following optimal control problems within a unified framework of function approximation and constrained optimization techniques. It allows for implementation of genetic algorithms for global optimization, multiobjective control, and utilization of parallel processing.

An optimal path following algorithm for control of multi-body systems is developed, and is tested for minimum-time-energy control of two-link manipulator. The path following optimal control problem is reduced to a constrained optimization problem by introducing a parameter set in order to approximate the control inputs. In the open-loop case, controls are regarded as the function of the time variable, t, while in the closed-loop case they are considered as functions of the state variables. Consequently, the original open-loop optimal control problem is reduced to a nonlinear programming case and the closed-loop problem is reduced a standard optimization problem. Once the problem is set up, the goal function and associated constraints are evaluated and the global optimum of the constrained optimization problem is found. In the case of closed-loop control, identification of the control law based on the input/output relations obtained from the solution of the open-loop optimal control problem needs to be done. The methodology was tested by simulating a model of a robot with one revolute and one prismatic joint with both open-loop and closed-loop optimal control.

Li, Tso, and Zhang [Li98] have proposed a dual-model-based structure for uncertainty attenuation in the trajectory-tracking control of robot manipulators. The traditional advanced control schemes are model based in the sense that a dynamic model, described either explicitly as a mathematical equation or implicitly as an input-output relation, is used in their design or final implementation, as in [Slo87, Tsu90]. The design of this type of model based control (MBC) is usually achieved in two steps: an *a priori* estimated dynamic model of the plant is first implemented in the control structure to linearize plant dynamics, and another controller is then designed for the resultant simplified dynamics to achieve a desired performance.

The application of MBC is however hindered by a practical problem: how to determine the actual dynamics of a complicated, highly non-linear, and highly coupled robot manipulator. Some of these issues can be resolved by modeling the manipulator links as rigid bodies, however the problem of varying payloads, unknown effects of friction, external disturbances, remain. The improvements to deal with such issues have come in two broadly generalized areas: adaptive MBC and robust MBC. In the adaptive MBC structures, an adaptation mechanism is applied to capture the payload variations and compensate for the unmodeled dynamics and external disturbances. The robust MBC approaches employ a fixed-structure control algorithm in their configurations to generate a robust performance with low sensitivity to modeling uncertainties. Compared with adaptive approaches, the robust approaches are simpler to implement and require less computational load.

The authors assert, based on the current state of control technology, that an improved controller scheme should lead to satisfying the following features: the control performance is insensitive to modeling errors and external disturbances; no high feedback gains are utilized in the design; the modification structure is simple and the performance analysis straightforward. In lieu of this assertion, the authors have presented a dual-model-based control (DMBC) structure for the trajectory-tracking control. This structure consists of two prime elements: a basic control structure and an uncertainty attenuation compensator.

The basic control algorithm is first applied to modify the robot dynamics so that it becomes an open-loop stable system. The authors then select the robust approach in the form of internal model control (IMC) structure to further compensate for uncertainties. It is stated that this DMBC approach is an enhanced scheme of the conventional robot MBC algorithm, with the IMC outer loop added as a complementary structure. Simulation studies were performed to demonstrate the effectiveness of this method. A 2-link (2R) manipulator was used with the basic control structure as well as with the additional compensation structure. The results showed that the effects of modeling uncertainties (comprised of parameter errors and unaccounted frictional effects) were significantly reduced by the DMBC approach. Similar results were obtained while studying the effects or external disturbances. It is noted however, that before the disturbance errors were introduced, both systems had tracking errors due to quantization error in the simulation. Of importance is the fact that the DMBC errors are 10 times those of the single controller. This is attributed to the side effects of the filter dynamics, which can be reduced by choosing a smaller filter time constant at the expense of higher sampling speed.

In another approach combining path planning and tracking control of a robot manipulator, Park and Park [Par97] have suggested the use of neural networks. This approach is used to deal with the presence of system uncertainty and obstacles and employs the use of neural networks combined with a conventional feedback controller. The path planner provides not only the (sub) optimal trajectory for a given cost function but also the configurations of the robot manipulator along the path by considering robot dynamics. The path is adjusted to avoid singular points and obstacles. An additional neural controller compensates for the tracking errors caused by uncertainty and disturbances, which provides robustness with a good tracking performance.

Amin and Ahtiwash [Ami96] have proposed another neural network based scheme. The authors employ two neuro-controllers utilizing the back propagation algorithm for robot tracking performance: the Inverse Neuro-Controller and the Neuro-Emulator Neuro-Controller schemes. For a given task of moving a robot from an initial rest position to a final specified position in minimum time, the profiles for position and velocity are determined. The results indicate that the two neuro-controllers allowed smooth and accurate motion of the manipulator.

CHAPTER 4
PATH TRACKING METHODOLOGIES

The following material presents an investigation into the development of path tracking methodologies and their application to autonomous robots, serial manipulators, and parallel platforms. The underlying assumption throughout the work is that the desired continuous path is known as a function of time, and if only discrete points along the path are known, a continuous approximation can be generated.

These approaches were developed in part from inferences made from preliminary attempts at path tracking techniques that are presented in Appendix A. The results of the simulations based on these earlier approaches, while not being very promising, did provide insightful information. It appeared that the mechanism observed a chaotic behavior and did not converge to a solution. Although the accelerations and velocities were quite high and discontinuous, it was anticipated that the error could be reduced and the solution would converge. This unexpected behavior was quite peculiar and lead to some conjectures:

- The behavior could have been the result of the method being based purely on a proportional controller type algorithm i.e., correction was attempted purely for error, and with no damping in the system, stability could have been an issue.
- Another reason for this type of behavior could have been the interaction of the links of the manipulator with each other. A solution to this problem was testing the algorithm for an autonomous robot case so that there is no interaction amongst the links.

Based on these observations, the basis of path tracking is developed first, followed by path tracking for autonomous robots, serial manipulators, and parallel platforms. The developed algorithms were implemented in simulation and their results follow each section.

Screw Theory Based Path Tracking

The problem of a rigid body following a path with a desired pose (position and orientation) is considered first. Consider the situation described in Figure 4.1. The desired path of the rigid body B is given by the timed path of point O, and the angle θ(t), which is the angle between the x-axis of a coordinate system fixed in rigid body B and a fixed reference. It is assumed that at some time t_o, the current pose of body B is given by $B_c(t_o)$. It is also assumed in this case that only pose information is available.

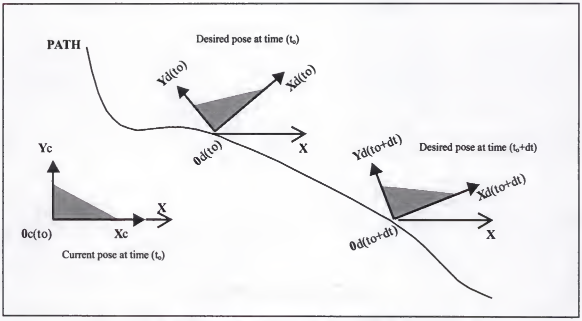

PATH
Desired pose at time (t_o)
Yd(to)
Xd(to)
X
0d(to)
Yc
Yd(to+dt)
Desired pose at time (t_o+dt)
Xd(to+dt)
X
0d(to+dt)
X
0c(to)
Xc
Current pose at time (t_o)

Figure 4.1: A rigid body following a path with a prescribed pose.

The problem is to devise a technique so that the error between the current and the desired poses is either eliminated or minimized to within a certain threshold. Chasles' Theorem [Cha30] will be used as the basis of undertaking this problem. Chasles' Theorem states that given the two positions of a rigid body, there is a unique screw displacement that moves the body from one position to the other. This result will be used to investigate a variety of techniques to solve the problem.

The solution is first attempted without regard to time, i.e., time is of no consequence in the desired displacement. From Figure 4.1, the following positions are considered:

- The current pose at time t_o, and
- The desired pose at time t_o

Using these two positions, Chasles theorem ensures that there is a unique screw displacement that moves the rigid body B from the current position to its desired position. Let this screw displacement be denoted by the screw, $\$_{c(t_o)\to d(t_o)}$. Further let $\Delta\theta$ be the rotation angle and $h = \Delta s / \Delta\theta$ be the pitch associated with it, where Δs is the linear translation along the screw axis. Finally, assume that the screw displacement will take place in a time interval Δt. Then, assuming that the displacement takes place at a constant angular velocity, the angular velocity associated with the rigid body (or the end effector) is given by:

$$\omega = \frac{\Delta\theta}{\Delta t} \tag{4.1}$$

Further, the constant velocity state associated with the rigid body B (or the end effector, in case of a serial manipulator) is given by:

$$\vec{V} = \omega \, \$_{c(t_o)\to d(t_o)} \tag{4.2}$$

The final result of this path tracking technique is that at time $t_o+\Delta t$, the rigid body B will have the desired position at time t_o. Thus, timewise, the tracking will be behind the desired tracking by an interval of Δt. Alternatively, it can be asserted that body B will track the desired path with a lag.

Extending the above case and considering time as a factor, a rather simple way to avoid the lag or time delay of the previous technique is as follows:

- The positions to be considered are: The current pose at time to and the desired pose at $t_o+\Delta t$.
- To require that the resulting screw displacement will take place in the time period Δt.

The first condition will determine a different screw, $\$_{c(t_o)\rightarrow d(t_o+\Delta t)}$, a different rotation angle $\Delta\theta$, and a different pitch $h=\Delta s/\Delta\theta$. The second condition would yield that the constant angular velocity, associated with the screw is,

$$\omega = \Delta\theta / \Delta t \tag{4.3}$$

and the constant velocity state of the rigid body (or the end-effector for a serial chain) is given by

$$\vec{V} = \omega \$_{c(t_o)\rightarrow d(t_o+\Delta t)} \tag{4.4}$$

This technique ensures that at time $t_o+\Delta t$, both the current and the desired positions coincide. Whatever the employed technique, if the rigid body B is the end-effector of a non-redundant spatial serial chain, *for example*, the corresponding joint velocities can be computed as follows (Figure 4.2):

$${}_0\omega_1 {}^0\$^1 + {}_1\omega_2 {}^1\$^2 + {}_2\omega_3 {}^2\$^3 + {}_3\omega_4 {}^3\$^4 + {}_4\omega_5 {}^4\$^5 + {}_5\omega_6 {}^5\$^6 = \vec{V} \tag{4.5}$$

A closer look at these techniques obviates the fact that these techniques would produce commands

- with discontinuous angular velocities and
- with discontinuous locations of screw axes.

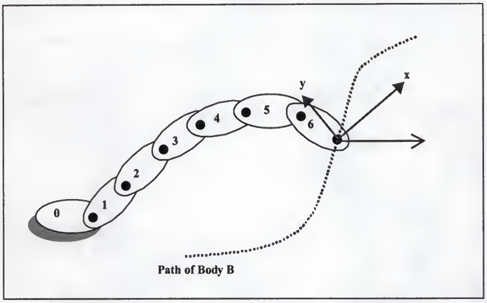


Figure 4.2: A serial manipulator traversing a prescribed path.

These discontinuities will produce infinite accelerations, and obviously undesirable characteristic for any robotic system. Additionally, there is another quite subtle source of a problem. It should be noted that in the case of a serial manipulator, Equation (4.5) can be used to calculate the joint velocities ${}_{i}\omega_{i+1}, (i = 0,1,2..,5)$. However, in the time interval Δt, there will be position changes that will affect the screws ${}^{i}\$^{i+1}$. Thus the joint velocities are only accurate at $t=t_o$. This however, might not be a major source of problem as the transient error may be reduced to an acceptable threshold by varying Δt (making it smaller).

Autonomous Robot / Vehicle

Considering Pose Information Only

Now, the problem of an autonomous robot following a path with a desired pose (position and orientation) is considered as shown in Figure 4.3. The desired time dependent path, $(x_d(t), y_d(t); \theta_d(t))$, is assumed to be specified. The current pose of the body at the instant t_0, is given as $(x_c(t_0), y_c(t_0); \theta_c(t_0))$ and the desired pose at time $(t_0 + \Delta t)$ is $(x_d(t_0 + \Delta t), y_d(t_0 + \Delta t); \theta_d(t_0 + \Delta t))$.

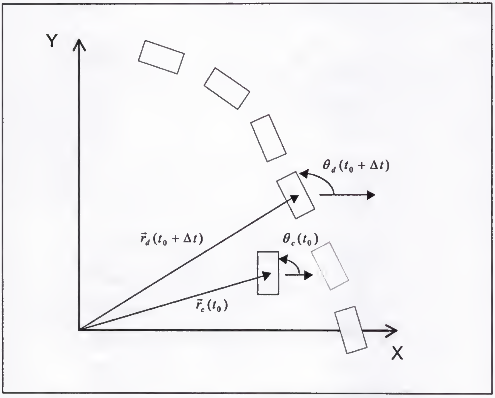


Figure 4.3: Autonomous robot path tracking with pose constraints.

Considering the position level only, the following desired velocity screw can be derived (from Equation 2. 39):

$$\vec{V} = \begin{bmatrix} 0 \\ 0 \\ \omega_z \\ \upsilon_x \\ \upsilon_y \\ 0 \end{bmatrix} \tag{4.6}$$

where,

$$\left.\begin{aligned} \omega_z &= \frac{\theta_d(t_0+\Delta t)-\theta_c(t_0)}{\Delta t} \\ \upsilon_x &= \frac{x_d(t_0+\Delta t)-x_c(t_0)}{\Delta t} \\ \upsilon_y &= \frac{y_d(t_0+\Delta t)-y_c(t_0)}{\Delta t} \end{aligned}\right\} \tag{4.7}$$

The resultant motion due to this screw will ensure that at time $(t_0+\Delta t)$ the position and orientation (pose) will coincide with the desired pose. It should be pointed out here that this method assumes that all velocity screws are possible, which may not be entirely feasible due to vehicular movement constraints. It should be noted that at the pose level, the commanded motion is continuous at the position level and is a linear approximation of the trajectory.

Considering Pose and Velocity Information

In this approach, the desired velocity state is also specified in addition to the pose information. *Initially*, it is assumed that current pose and velocity information is known. As shown in Figure 4.4, the current state is defined by:

$$\left.\begin{matrix} x_c(t_0), y_c(t_0); \theta_c(t_0) \\ \upsilon_{xc}(t_0), \upsilon_{yc}(t_0); \omega_c(t_0) \end{matrix}\right\}$$

and the desired state at instant $(t_0+\Delta t)$ is:

$$\left.\begin{array}{l} x_d(t_0+\Delta t), y_d(t_0+\Delta t); \theta_d(t_0+\Delta t) \\ \upsilon_{xd}(t_0+\Delta t), \upsilon_{yd}(t_0+\Delta t); \omega_d(t_0+\Delta t) \end{array}\right\}$$

Using the pose information, the velocity screw, $\vec{V}$ (4.6), as done previously, can be calculated as :

$$\left.\begin{array}{l} \omega_z = \dfrac{\theta_d(t_0+\Delta t)-\theta_c(t_0)}{\Delta t} \\ \upsilon_x = \dfrac{x_d(t_0+\Delta t)-x_c(t_0)}{\Delta t} \\ \upsilon_y = \dfrac{y_d(t_0+\Delta t)-y_c(t_0)}{\Delta t} \end{array}\right\} \qquad (4.8)$$

Further, from the velocity information, it is possible to determine the following:

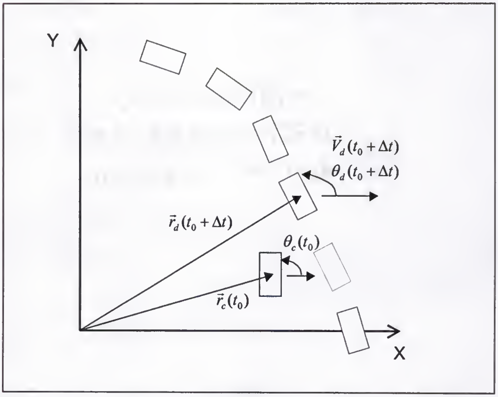


Figure 4.4: Autonomous robot path tracking with pose and velocity information.

$$\left.\begin{aligned} \alpha_z &= \frac{\omega_d(t_0+\Delta t)-\omega_c(t_0)}{\Delta t} \\ a_x &= \frac{\upsilon_{xd}(t_0+\Delta t)-\upsilon_{xc}(t_0)}{\Delta t} \\ a_y &= \frac{\upsilon_{yd}(t_0+\Delta t)-\upsilon_{yc}(t_0)}{\Delta t} \end{aligned}\right\} \qquad (4.9)$$

This information, coupled with the velocity state at t_0, will provide a reduced acceleration screw $\vec{A}_R$. However, it is important to note that determining how the two screws, velocity and acceleration, will affect each other is an issue that needs to be addressed.

The problem is now set up differently in a way conducive to determining the common action of two screws. Assume that the initial or current pose information (state) of the body is known. Consider a rigid body B and let $(\upsilon_{x0},\upsilon_{y0};\omega_0)$ denote the initial or current velocity of the body and let $(\upsilon_{x0c},\upsilon_{y0c};\omega_{0c})$ represent the (to be determined) required command velocity parameters. Also let $(a_x,a_y;\alpha)$ denote the acceleration state. Then after a time step, Δt, the final velocity and position is given by:

Velocity:

$$\left.\begin{aligned} \omega_f &= \omega_0+\alpha\Delta t \\ \upsilon_{xf} &= \upsilon_{x0}+a_x\Delta t \\ \upsilon_{yf} &= \upsilon_{y0}+a_y\Delta t \end{aligned}\right\} \qquad (4.10\ a,b,c)$$

Position:

$$\left.\begin{aligned} \theta_d &= \theta_0+\omega_0\Delta t+\tfrac{1}{2}\alpha\Delta t^2 \\ x_d &= x_c+\upsilon_{x0}\Delta t+\tfrac{1}{2}a_x\Delta t^2 \\ y_d &= y_c+\upsilon_{y0}\Delta t+\tfrac{1}{2}a_y\Delta t^2 \end{aligned}\right\} \qquad (4.11\ a,b,c)$$

These equations above are the algebraic equivalent of the common action of a velocity and an acceleration screw. It will now be shown how the command parameters $(\upsilon_{x0c}, \upsilon_{y0c}; \omega_{0c})$ will be calculated. Of course, the calculated command parameters may differ from the current velocity parameters, and any control system that is applied will act to minimize this difference. Assuming that the initial pose $(x_0, y_0; \theta_0)$ is known and the final pose $(x_f, y_f; \theta_f)$, the final velocity $(\upsilon_{x0f}, \upsilon_{yf}; \omega_f)$, and the time interval Δt are specified, Equations (4.10) and (4.11) above are used to solve for ω_{0c} and α as follows:

From Equation (4.10)a we have,

$$\alpha = \frac{\omega_f - \omega_{0c}}{\Delta t} \tag{4.12}$$

substituting this in Equation (4.11)a gives,

$$\theta_f = \theta_0 + \omega_{0c}\Delta t + \tfrac{1}{2}\left(\frac{\omega_f - \omega_{0c}}{\Delta t}\right)\Delta t^2$$

rearranging, we obtain

$$\begin{aligned}\theta_f - \theta_0 &= \omega_{0c}\Delta t + \tfrac{1}{2}\omega_f\Delta t - \tfrac{1}{2}\omega_{0c}\Delta t \\ &= \tfrac{1}{2}\omega_{0c}\Delta t + \tfrac{1}{2}\omega_f\Delta t\end{aligned}$$

and finally solving for ω_{0c},

$$\omega_{0c} = \frac{2(\theta_f - \theta_0)}{\Delta t} - \omega_f \tag{4.13}$$

where ω_{0c} is the velocity for the time step Δt that would satisfy the pose and velocity requirements at $t + \Delta t$.

To solve for α, substituting Equation (4.13) in Equation (4.12):

$$\alpha = \frac{\omega_f}{\Delta t} - \frac{1}{\Delta t}\left(\frac{2(\theta_f - \theta_0)}{\Delta t} - \omega_f\right)$$

$$\alpha = \frac{1}{\Delta t}\left(\omega_f - \frac{2(\theta_f - \theta_0)}{\Delta t} + \omega_f\right)$$

$$\alpha = \frac{2}{\Delta t}\left(\omega_f - \frac{(\theta_f - \theta_0)}{\Delta t}\right) \tag{4.14}$$

The linear velocity and acceleration is calculated by following a similar procedure. Considering Equations (4.10) b,c and Equations (4.11) b,c respectively:

$$\left.\begin{aligned} a_x &= \frac{\upsilon_{xf} - \upsilon_{x0c}}{\Delta t} \\ a_y &= \frac{\upsilon_{yf} - \upsilon_{y0c}}{\Delta t} \end{aligned}\right\}$$

$$\left.\begin{aligned} x_d &= x_c + \upsilon_{x0c}\Delta t + \tfrac{1}{2}\left(\frac{\upsilon_{xf} - \upsilon_{x0c}}{\Delta t}\right)\Delta t^2 \\ y_d &= y_c + \upsilon_{y0c}\Delta t + \tfrac{1}{2}\left(\frac{\upsilon_{yf} - \upsilon_{y0c}}{\Delta t}\right)\Delta t^2 \end{aligned}\right\}$$

In a procedure analogous to the procedure for determining ω_{0c} and α above, the following unknowns are determined:

$$\upsilon_{x0c} = \frac{2(x_f - x_0)}{\Delta t} - \upsilon_{xf} \tag{4.15}$$

$$\upsilon_{y0c} = \frac{2(y_f - y_0)}{\Delta t} - \upsilon_{yf} \tag{4.16}$$

and

$$a_x = \frac{2}{\Delta t}\left(\upsilon_{xf} - \frac{(x_f - x_0)}{\Delta t}\right) \tag{4.17}$$

$$a_y = \frac{2}{\Delta t}\left(\upsilon_{yf} - \frac{(y_f - y_0)}{\Delta t}\right). \tag{4.18}$$

Once all the unknowns are determined, the desired velocity state and the acceleration state of the body are known:

$$\vec{V}_d = \begin{bmatrix} 0 \\ 0 \\ \omega_d \\ \upsilon_{xd} \\ \upsilon_{yd} \\ 0 \end{bmatrix} \quad , \quad \vec{\alpha} = \begin{bmatrix} 0 \\ 0 \\ \alpha \end{bmatrix} \quad , \quad \vec{a} = \begin{bmatrix} a_x \\ a_y \\ 0 \end{bmatrix} \tag{4.19}$$

and the reduced acceleration screw (the reduced acceleration state) is calculated as:

$$\vec{A}_R = \begin{bmatrix} \vec{\alpha} \\ \vec{a}_0 - \vec{\omega} \times \vec{\upsilon}_0 \end{bmatrix} \tag{4.20}$$

It should be noted that when pose and velocity information is used, the commanded motion is continuous at the pose and velocity level, which implies that the commanded motion is a quadratic approximation of the trajectory.

The results obtained from the analysis were employed for a simulation. The simulation was done in MATLAB®. The test cases and the results obtained are presented next.

The developed algorithm was tested with two cases. In both cases the desired path was a circle. The idea behind using a circular path was that for such a motion, the instantaneous screw axis(pole) would be the center of the circle and would provide a means of determining an error; which otherwise would be difficult as mixed angular and linear units are involved. Using Equations (4.6) and (4.7), the constant angular velocity is determined to meet the *pose* tracking criteria. From that information, the location of the pole or ISA is determined using

$$\vec{r}_p = \vec{r}_i + \frac{\vec{\omega} \times \vec{\upsilon}_i}{|\vec{\omega}|^2} \tag{4.21}$$

In the first instance (autonomous.m, Appendix B), the center of the circle was located at (1,0) and the starting point for the robot was(1.51, -.05,0). The results of the tracked path are shown in Figure 4.5a. The axes of the graphs represent the x and y directions of the plane. The 'o' legend in the center of the circle represents the pole locations of the instantaneous screw axis at each interval of time Δt. Figure 4.5b shows a zoomed-in version of the pole locus. It is interesting to note that the root locus is a circle too as opposed to being the center point of the circle. This discrepancy is attributed to using a difference formula for determining the angular and linear velocities. In the second instance (autonomous1.m, Appendix B), the circle was located at (1,0.5) and the initial position was specified as (1.5, 0.5). The results are shown in Figure 4.6a and 4.6b respectively.

In the second case when both the *pose and desired velocity* of the robot is specified, Equations (4.13) through (4.18), which require pose and velocity state information, are used to determine the angular velocities and constant angular

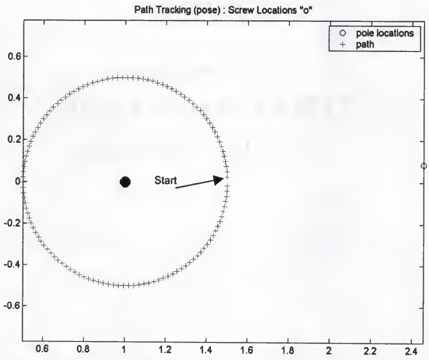


Figure 4.5a: Autonomous Robot: pose information only. Tracked path and pole locations.

accelerations to meet the *pose and velocity* tracking criteria. Again, the location of the poles or ISA's are determined using Equation (4.21).

Two paths were considered in the simulations. In the first instance (autonomous2a.m, Appendix B), the path was a circle with radius 0.5 centered at (1,0.5). In this case, the orientation was also variant, i.e., the desired orientation was set to be tangent to the circle at every instant. The starting position of the robot was set arbitrarily at (1.54, .45). Figure 4.7 shows the path tracked and the locus of poles. The first pole appears at quite a distance from the rest of the poles, due to the fact that the starting position of the robot is off from the desired position. Also shown in the figure is the position and orientation of the robot at each time period. The second case (autonomous3.m, Appendix B) that was simulated involved tracking a parabolic path. The starting point of the robot was set at (-0.45, 0.81). The desired orientation changed as a function of time by $(5 \times \Delta t)$. The results are shown in Figure 4.8.

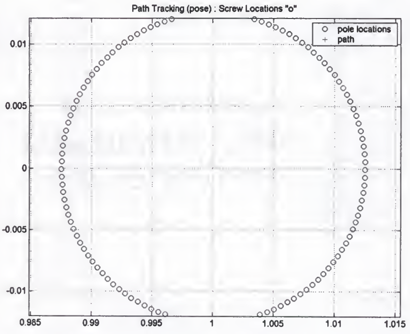


Figure 4.5b: Autonomous Robot: pose information only. Pole locations.

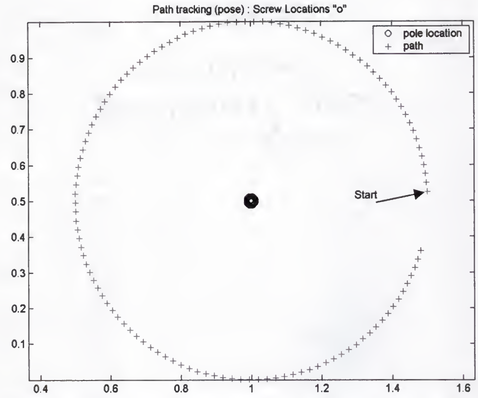


Figure 4.6a: Autonomous Robot: pose information only. Tracked path and pole locations.

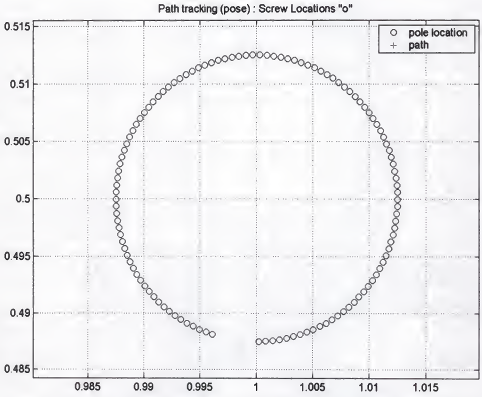


Figure 4.6 a: Autonomous Robot: pose information only. Pole locations.

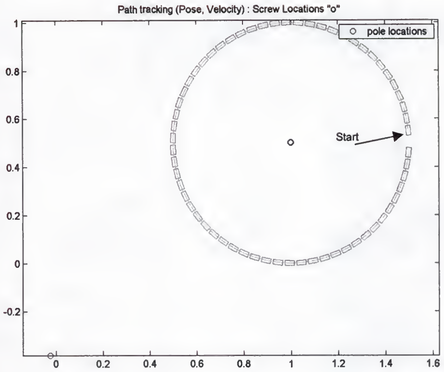


Figure 4.7: Autonomous robots, pose and velocity information. Poles, tracked path, and vehicle orientation (box).

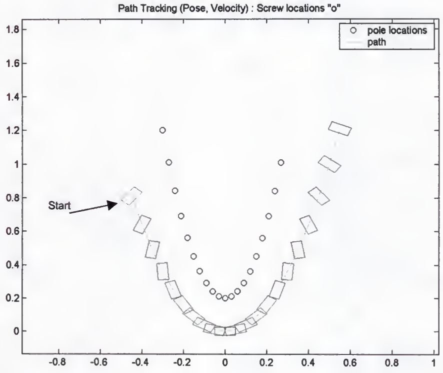


Figure 4.8: Autonomous robots, pose and velocity information. Poles, tracked path, and vehicle orientation (box).

3R Planar Manipulator

Considering Pose Information Only

A new approach to the tracking analysis for a 3R Manipulator is developed (as opposed to the approach discussed in Appendix A). Consider the 3R Planar Manipulator shown in Figure 4.9, with the pose information available, i.e., the current and desired pose states are known.

The approach utilizes the *inverse position analysis* of the manipulator to determine the current joint angles for position, θ_{ic}, and the joint angles for the desired position, θ_{id}. From the results of the analysis, the joint angular velocity is calculated by using the formula (Equation (4.7)):

$$\omega_i = \frac{\theta_{id}(t_0 + \Delta t) - \theta_{ic}(t_0)}{\Delta t} \tag{4.22}$$

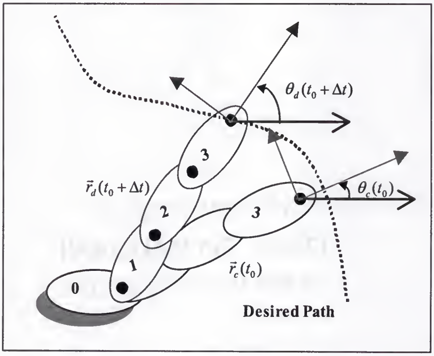


Figure 4.9: 3R Manipulator with pose information.

To determine the errors in the desired and commanded velocities, the desired velocity state is determined from the path of the manipulator. Also, from this desired velocity state, the joint angular velocities are calculated by using the following formula, the underlying assumption being that the path is continuous and differentiable:

$$ {}_0\omega_1 {}^0\$^1 + {}_1\omega_2 {}^1\$^2 + {}_2\omega_3 {}^2\$^3 = \vec{V} \tag{4.23}$$

$$[J]\vec{\omega} = \vec{V} \tag{4.24}$$

$$\vec{\omega} = [J]^{-1}\vec{V} \tag{4.25}$$

where, $[J]$ is the Jacobian of the manipulator, $\vec{\omega}$ is a vector defining the joint angular velocities of the manipulator, and $\vec{V}$ is a vector of the velocity state of the manipulator.

Path tracking of a 3R planar manipulator was simulated using the current and desired *pose information* using this methodology. A flowchart of the algorithm is shown in Figure 4.10. All the link lengths of the manipulator were set as unity. The desired motion of the manipulator was a circle specified by $(0.5 + r\cos(t), r\sin(t))$. The orientation was not varied. The algorithm (main4.m, Appendix B) uses the inverse position analysis to determine a set of joint angles. These joint angles are then used to compute the command angular velocities. To determine the effectiveness of the algorithm, the calculated velocity parameters (command angular velocities) are compared to the velocity parameters that are determined using infinitesimal motion screws (desired angular velocities). This provides a rough order of magnitude comparison and validation of the command screw. The algorithm computes the Jacobian of the manipulator, which coupled with the information about the velocity state, provides the desired joint angular velocities. The two values are compared to determine the error (Figure 4.10).

To illustrate how the algorithm works, an iteration of the procedure is shown from the algorithm main4.m. The current pose or state of the manipulator is given as

$$r_c = (2,1,0) \quad , \quad \phi_c = 0$$

where ϕ is the orientation of the end effector. The desired path and velocity for the manipulator is specified as (a circle with radius 0.2 and the center at (0.5, 0))

$$r_d = (0.5 + 0.2 * \cos(t+\Delta t), 0.2 * \sin(t+\Delta t), 0) \quad , \quad \phi_d = 0 \tag{4.26}$$

$$V_d = (-0.2 * \sin(t+\Delta t), 0.2 * \cos(t+\Delta t), 0). \tag{4.27}$$

Inverse position analysis is performed by the function reverse_position_3R at the current pose at time t and at the next desired pose at instant $t + \Delta t$. The inputs to the function are the link lengths of the manipulator and the current position of the end effector. The outputs of the function are the joint angles of the manipulator and the Jacobian matrix (as defined in Equation (2.49)). Using the joint angle information at instant t and $t + \Delta t$ the command velocity screw for each joint is generated using Equation (4.22) as

$$\omega_{command} = \frac{\theta_d - \theta_c}{\Delta t}. \tag{4.28}$$

The Jacobian matrix is used to determine the joint angular velocities as dictated by the desired velocity state, V_d, by using Equation (4.25) as

$$\omega_{calculated} = [J]^{-1} V_d. \tag{4.29}$$

The two joint velocities are then compared to verify that the generated command satisfies the desired velocity constraints. The results indicate that the command velocity is within the desired velocity range by an average of 0.006% with a standard deviation of 0.02 rad/sec.

Figure 4.11 shows the manipulator and the traversed path. Figures 4.12a and 4.12b, and Figures 4.13a and 4.13b show the actual and desired angular velocities and the errors respectively. The errors are high at the initial step as the algorithm attempts to jump to the desired position in a single step. This can be addressed by dividing the motion of the first step into smaller increments or by considering reducing the error in more than one step, for example, in $2\Delta t$ instead of Δt. It should also be noted that this step is application dependent; i.e., this issue is generally avoidable if the path is pre-planned and the initial starting point for the manipulator is known.

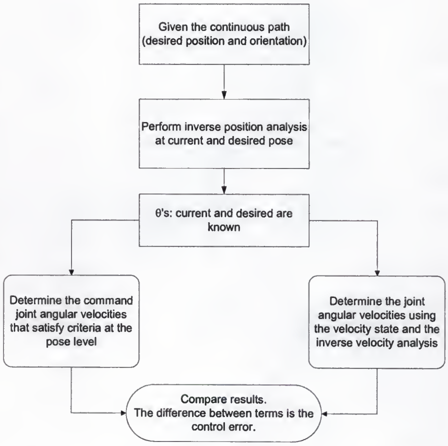

Given the continuous path (desired position and orientation)
Perform inverse position analysis at current and desired pose
θ's: current and desired are known
Determine the command joint angular velocities that satisfy criteria at the pose level
Determine the joint angular velocities using the velocity state and the inverse velocity analysis
Compare results. The difference between terms is the control error.

Figure 4.10: Algorithm summary for 3R Manipulator path tracking using pose information.

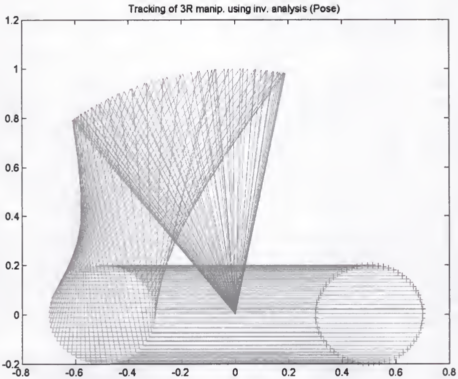


Figure 4.11: 3R Manipulator path tracking with pose information. Manipulator and the traversed path.

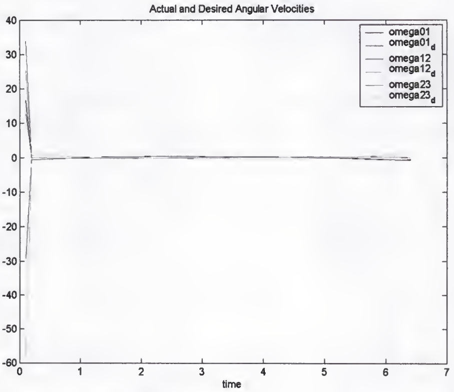


Figure 4.12a: 3R Manipulator path tracking with pose information. Actual and desired angular velocities.

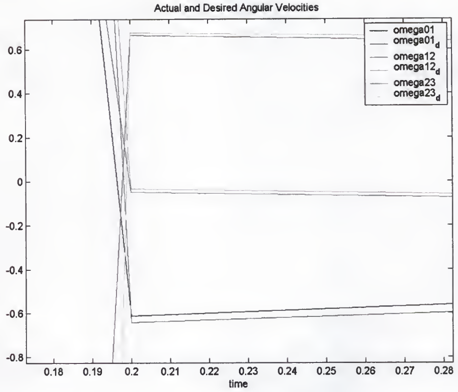


Figure 4.12b: 3R Manipulator path tracking with pose information. Actual and desired angular velocities (magnified view).

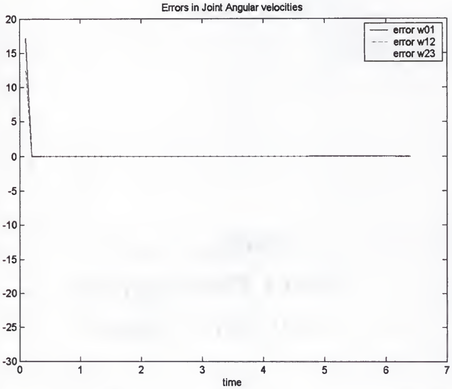


Figure 4.13a: 3R Manipulator path tracking with pose information. Errors in joint angular velocities.

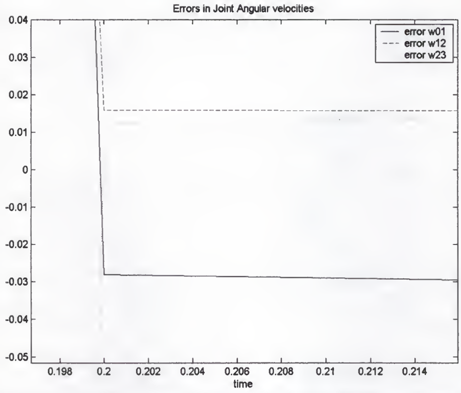


Figure 4.13b: 3R Manipulator path tracking with pose information. Errors in joint angular velocities (magnified view).

<u>Considering Pose and Velocity Information</u>

The above analysis is further extended to consider the case where the desired velocity state is also specified as a constraint (Figure 4.14). The analysis is done along the same lines as for the autonomous case in the previous section whereby the common action of the acceleration and velocity screw is determined. In addition to determining the joint angular velocities by using Equation (4.13)

$$\omega_i = \frac{2(\theta_d - \theta_c)}{\Delta t} - \omega_d \tag{4.30}$$

the joint angular accelerations are generated from (Equation (4.14):

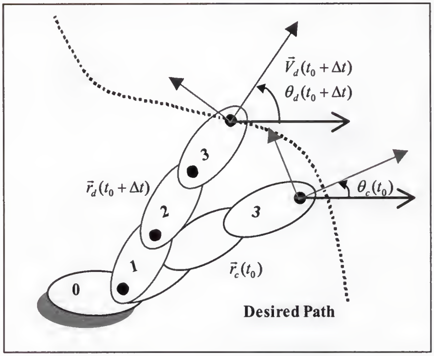


Figure 4.14: 3R Manipulator with pose and desired velocity

$$\alpha_i = \frac{2}{\Delta t}\left(\omega_d - \frac{(\theta_d(t_0+\Delta t)-\theta_c(t_0))}{\Delta t}\right) \tag{4.31}$$

In a way analogous to the pervious section, in order to determine the errors in the desired and calculated accelerations, the desired acceleration state is determined from the velocity state of the manipulator (again, the assumption being that the velocity function is continuous and differentiable). From the desired acceleration state, the joint angular accelerations are determined as follows.

The equation determining the reduced acceleration state of a manipulator is:

$$\vec{A}_R \quad = \quad \left\{ \begin{matrix} {}_0\dot{\omega}_1{}^0\$^1 + {}_1\dot{\omega}_2{}^1\$^2 + {}_2\dot{\omega}_3{}^2\$^3 + \\ \left[{}_0\omega_1{}^0\$^1 \quad {}_1\omega_2{}^1\$^2 + {}_2\omega_3{}^2\$^3\right] + \\ \left[{}_1\omega_2{}^1\$^2 \quad {}_2\omega_3{}^2\$^3\right] \end{matrix} \right\} \tag{4.32}$$

where ω's represent the joint velocities, the $\dot{\omega}$'s represent the unknown joint accelerations, and \$ are the screws that comprise the Jacobian. Once the reduced acceleration state is known, the joint angular accelerations are determined:

The above equation can be written as

$$\vec{A}_R = {}_0\dot{\omega}_1{}^0\$^1 + {}_1\dot{\omega}_2{}^1\$^2 + {}_2\dot{\omega}_3{}^2\$^3 + \left[{}_0\omega_1{}^0\$^1 \quad {}_1\omega_2{}^1\$^2 + {}_2\omega_3{}^2\$^3\right] + \left[{}_1\omega_2{}^1\$^2 \quad {}_2\omega_3{}^2\$^3\right] \tag{4.33}$$

which in turn can be written as:

$$[J]\,\dot{\omega} + [L] = \vec{A}_R \tag{4.34}$$

$$[J]\,\dot{\omega} = \vec{A}_R - [L] \tag{4.35}$$

and finally solving for the acceleration,

$$\dot{\omega} = [J]^{-1}\,[\,\vec{A}_R - [L]\,] \tag{4.36}$$

where $[L]$ is the term containing all the Lie products.

In this case, the path tracking of a 3R manipulator was simulated with the desired velocity added as an additional constraint. The algorithm generates the command velocities and accelerations by using the position and desired velocity information. For comparison, the algorithm calculates desired acceleration state (from the desired velocity) and determines the desired joint accelerations. These are compared to the command velocities and accelerations to determine how closely they follow. The algorithm flowchart is shown in Figure 4.15.

As in the previous case, the link lengths of the manipulator are set as unity. Two different cases were considered. In the first case (Case I), the path to be tracked was the same as the one considered for the pose-only case (main54.m, Appendix B). This was done so that the results could be compared. The path simulation is shown in Figure 4.16. Figures 4.17 through 4.20 show the angular velocities and accelerations and the errors. A second case (Case II) was considered (main5a.m, Appendix B) in which the path was kept the same but the orientation of the end effector was varied as it traversed the path. This case is illustrated in Figures 4.21 through 4.25.

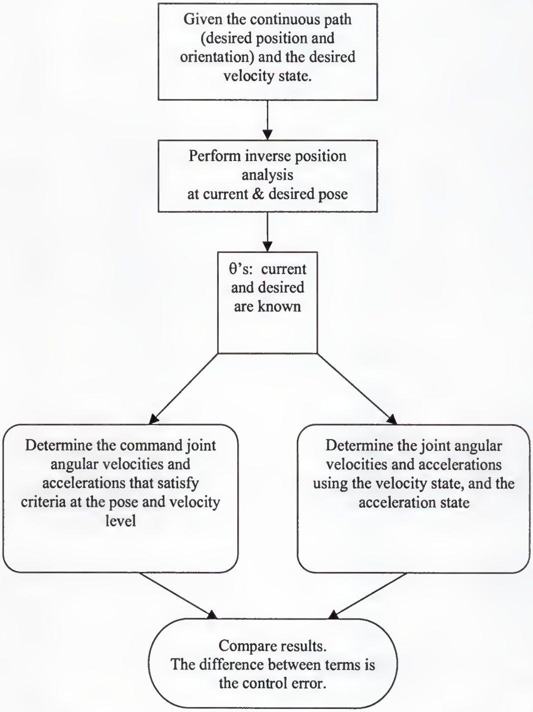

Given the continuous path (desired position and orientation) and the desired velocity state.
Perform inverse position analysis at current & desired pose
θ's: current and desired are known
Determine the command joint angular velocities and accelerations that satisfy criteria at the pose and velocity level
Determine the joint angular velocities and accelerations using the velocity state, and the acceleration state
Compare results. The difference between terms is the control error.

Figure 4.15: Algorithm summary for 3R Manipulator path tracking using pose and desired velocity information.

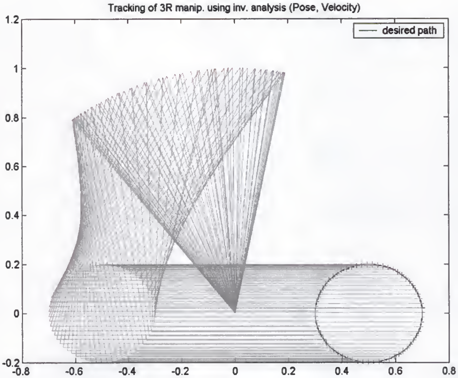


Figure 4.16: 3R Manipulator path tracking, using pose & desired velocity information. Motion of the manipulator and the traversed path. Case I.

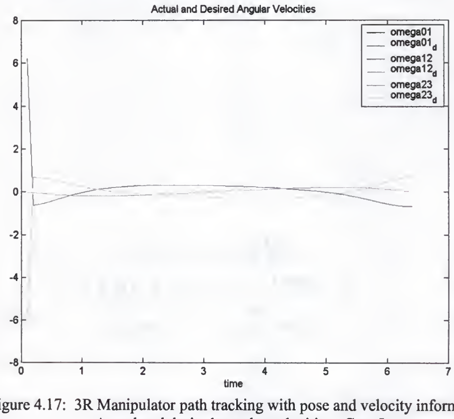


Figure 4.17: 3R Manipulator path tracking with pose and velocity information. Actual and desired angular velocities. Case I.

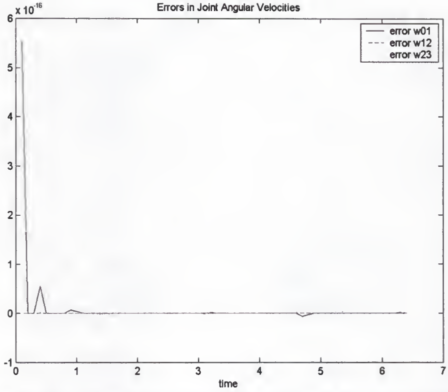


Figure 4.18: 3R Manipulator path tracking with pose and velocity information. Errors in joint angular velocities. Case I.

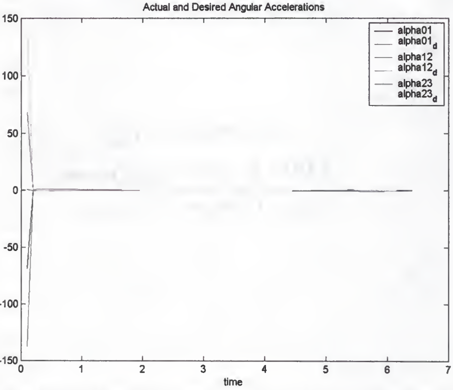


Figure 4.19: 3R Manipulator path tracking with pose and velocity information. Actual and desired angular accelerations. Case I.

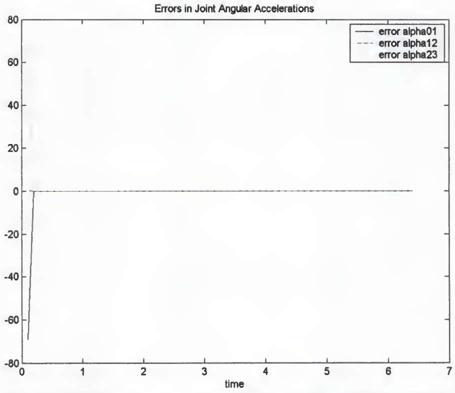


Figure 4.20: 3R Manipulator path tracking with pose and velocity information. Errors in joint angular accelerations. Case I.

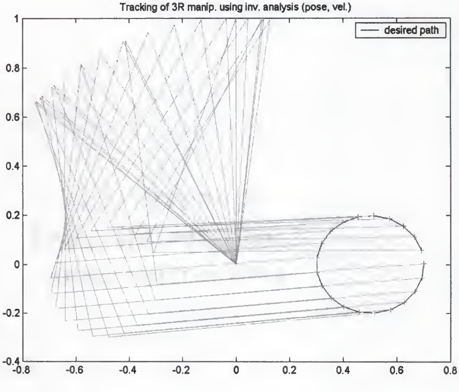


Figure 4.21: 3R Manipulator path tracking, using pose & desired velocity information. Motion of the manipulator and the traversed path. Case II.

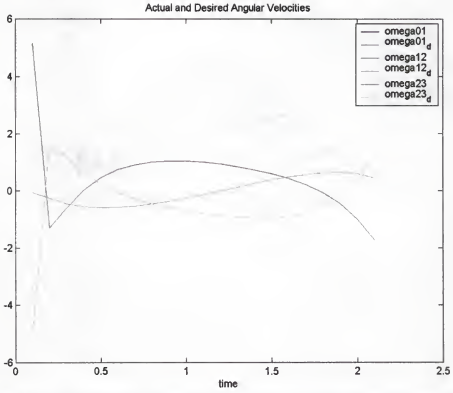


Figure 4.22: 3R Manipulator path tracking with pose and velocity information. Actual and desired angular velocities. Case II.

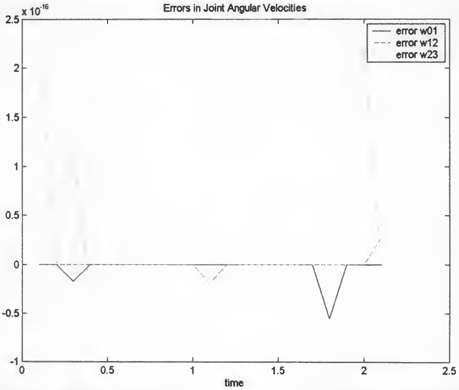


Figure 4.23: 3R Manipulator path tracking with pose and velocity information. Errors in joint angular velocities. Case II.

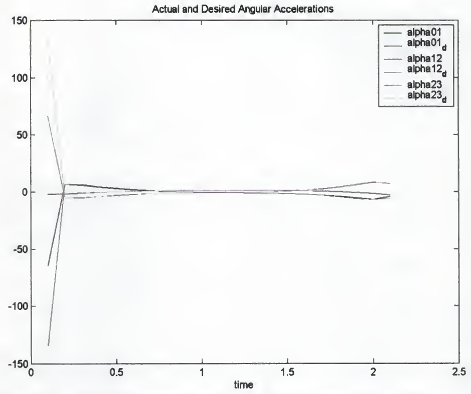


Figure 4.24: 3R Manipulator path tracking with pose and velocity information. Actual and desired angular accelerations. Case II.

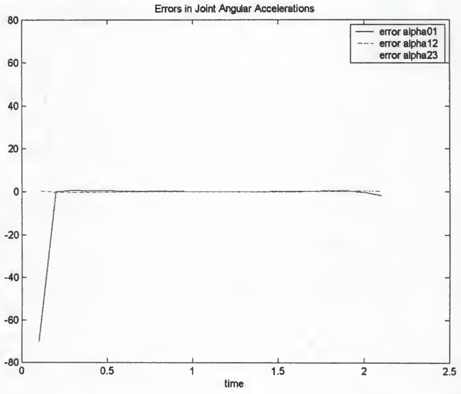


Figure 4.25: 3R Manipulator path tracking with pose and velocity information. Errors in joint angular accelerations. Case II.

Parallel Manipulators

Parallel mechanism based approaches to manipulator design have grown in popularity in the past decade or so. Mechanisms for a variety of applications including robotics, machine tools, measurement machines, and sensors have been developed based on parallel devices.

A parallel mechanism or a platform is defined as any mechanical device that has legs (connectors) that connect a moving platform to a base. This type of mechanism possesses desirable characteristics of high accuracy, high payload-to-weight ratio, and good static stability. The geometrically simplest parallel mechanism has the structure of an octahedron and it is designated as a "3-3 platform" since there are three connecting points on the base and the top platform respectively. This geometry while being simple, has a serious mechanical disadvantage as it is very difficult to design concentric ball and socket joints at each of the double connection points on the octahedron without mechanical interference. To avoid the shortcoming of this platform, platforms with different geometry have been developed, some of which are the 6-3 platform [Ste65], the 6-6 platform (hexapod), and the special 6-6 platform [Gri93].

To control these mechanisms it is necessary to determine accurate compliance models. These kinematic models can be obtained if the position and orientation of the moving platform is known relative to the base. This is known as the inverse kinematic analysis for the system. The complement of the inverse analysis is the forward analysis, which entails determining the position of the moving platform with respect to the base given the six connector lengths. For a general 6-6 platform, the closed form forward analysis equation is a 40th degree polynomial [Rag93], which is computationally impractical for real time control. On the other hand, the forward solution of a 3-3 platform involves solving an eighth degree polynomial in a single variable. The special 6-6 platform provides combines the benefits of both of these platforms. These platforms allow for the simple analysis of the 3-3 with an eighth degree polynomial, and allow for the general benefits of the 6-6 by eliminating mechanical interference.

Based on the combination of these advantages, the special 6-6 platform has been a subject of research at CIMAR for a variety of applications; among them is the development of a force-feedback end-effector, as well as the study of the platform as possible machine tool for milling applications [Abb00]. Currently, the platform is being used as the basis for development of a Weight Compensating Parallel Manipulator (spatial jack) to handle large payloads.

Tracking Methodology

The path tracking methodology developed for parallel manipulators involves the use of the inverse position and the inverse velocity analysis. The length and the velocity of each connector are then determined.

The initial attempt to solve this problem was by assuming an acceleration profile that was linear, i.e. the variation in the acceleration was linear (Appendix A). The results obtained from this analysis indicated that this assumption would not yield a result that would satisfy the position, velocity, and acceleration specifications at the end of each time step. To overcome this problem a different approach was attempted which is discussed below.

To meet the path tracking criteria, an approximation was required that would produce the desired motion performance at the beginning and end of each step. Since the criteria for motion planning were assumed to be position, velocity, and acceleration, these criteria involve six conditions at the beginning and end of each step. These conditions are the initial and final position, velocity, and acceleration of the actuators, respectively. Thus the approximation is based on a fifth order polynomial that has 6 constants. The 6 constants match the number of conditions required to satisfy exactly the position, velocity, and acceleration state at the beginning and end of a time step.

Consider the polynomial

$$s(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \tag{4.37}$$

differentiating with respect to time

$$s'(t) = a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4 \tag{4.38}$$

and again

$$s''(t) = 2a_2 + 6a_3 t + 12a_4 t^2 + 20a_5 t^3 . \tag{4.39}$$

For $t = 0$, and

$$s(0) = s_0$$

$$s'(0) = s_0'$$

$$s''(0) = s_0''$$

substituting in Equations (4.37),(4.38),and (4.39), yields

$$s_0 = a_0 \tag{4.40}$$

$$s_0' = a_1 \tag{4.41}$$

$$s_0'' = 2a_2 \text{ or} \tag{4.42}$$

$$a_2 = \frac{s_0''}{2} . \tag{4.43}$$

Substituting the known coefficients above in Equations (4.37),(4.38),and (4.39), we obtain

$$s(t) = s_0 + s_0' t + \tfrac{1}{2} s_0'' t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \tag{4.44}$$

$$s'(t) = s_0' + s_0'' t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4 \tag{4.45}$$

$$s''(t) = s_0'' + 6a_3 t + 12a_4 t^2 + 20a_5 t^3. \tag{4.46}$$

At the end of the time step Δt, the first, second and third order equations become

$$s_f = s_0 + s_0'\Delta t + \tfrac{1}{2}s_0''\Delta t^2 + a_3\Delta t^3 + a_4\Delta t^4 + a_5\Delta t^5$$

$$s_f' = s_0' + s_0''\Delta t + 3a_3\Delta t^2 + 4a_4\Delta t^3 + 5a_5\Delta t^4$$

$$s_f'' = s_0'' + 6a_3\Delta t + 12a_4\Delta t^2 + 20a_5\Delta t^3.$$

Rewriting the equations above and substituting δ for Δt for the sake of clarity,

$$s_f = s_0 + s_0'\delta + \tfrac{1}{2}s_0''\delta^2 + a_3\delta^3 + a_4\delta^4 + a_5\delta^5 \tag{4.47}$$

$$s_f' = s_0' + s_0''\delta + 3a_3\delta^2 + 4a_4\delta^3 + 5a_5\delta^4 \tag{4.48}$$

$$s_f'' = s_0'' + 6a_3\delta + 12a_4\delta^2 + 20a_5\delta^3. \tag{4.49}$$

The equations above have three unknowns, a_3, a_4, and a_5. These equations were solved using Maple®, and yielded the constants as

$$a_3 = \frac{1}{2}\left(\frac{7s_0''\delta^2 + 20s_f - 20s_0 - 12s_0'\delta + s_f''\delta^2 - 8s_f'\delta}{\delta^3}\right) \tag{4.50}$$

$$a_4 = \frac{6s_0''\delta^2 + 15s_f - 15s_0 - 8s_0'\delta + s_f''\delta^2 - 7s_f'\delta}{\delta^4} \tag{4.51}$$

$$a_5 = \frac{1}{2}\left(\frac{5s_0''\delta^2 + 12s_f - 12s_0 - 6s_0'\delta + s_f''\delta^2 - 6s_f'\delta}{\delta^5}\right). \tag{4.52}$$

Once all the coefficients $(a_1, a_2, a_3, a_4, a_5)$ are known, Equations (4.37), (4.38), and (4.39) can be used for generating commands that satisfy the desired conditions for position, velocity, and acceleration.

6-6 Parallel Manipulator

The parallel mechanism considered as an example is the special 6-6 parallel platform. The reason for considering this platform is the availability of a design methodology for designing such a platform to suit a particular application. The parametric methodology developed by Abbasi et al. [Abb00] is used and is concisely presented here.

Generally, when designing a platform, the anticipated application must be considered. The application defines the loading conditions, workspace geometry, and dexterity, based on which the platform parameters are optimized. The number of design variables for the special 6-6 platform is many. To examine various designs in a proficient manner a methodology has been developed that reduces the design variables.

The platform kinematics can be described parametrically by defining two triangles, the base and the top, and six side pivot connections. An equilateral triangle is used as provides symmetric stiffness and response over the workspace. The equilateral triangle can be defined by one parameter, the side of the triangle. The top triangle is related to the base triangle by the parameter *tsidescale*. The parameters bscale and tscale define the base and top side pivot points, respectively. Another parameter, *hscale*, is used to define the perpendicular distance between the top and the base of the platform. This parameter is also a function of SIDE. If *hscale* is equal

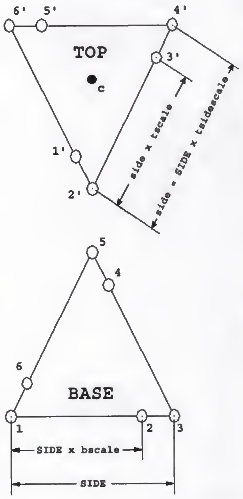


Figure 4.26: Design parameterization of the special 6-6 platform.

to 1, the distance between the top and the base of the platform is equal to SIDE, and a variation of this parameter would vary the height of the platform accordingly.

The special 6-6 parallel mechanism requires that the pivot connection lie on the line that contains the two vertex connections. For example, in Figure 4.26, side pivot connection 2 must lie on the same line that connects 1 and 3. On the base, if *bscale* is less than 1, then the side pivot lies between the two vertex connections and vice versa. The same is valid for *tscale* with respect to the top.

A given platform may have a large kinematic workspace, but its stability over the workspace may vary enough to have affect on its performance. The parameter SIDE allows the mechanism to be scaled up or down. The other parameters define the kinematic workspace and variation of static stability across the workspace. This methodology allows finding suitable parameters that define a platform that is well behaved over the workspace.

The inverse displacement analysis is useful in developing the expressions for the dynamic state of the system. The platform location must be specified for a particular task. The displacement and orientation of each of the connectors that actuate the platform can then be determined once the platform's motion time history has been specified. From the motion specification, the velocity and the acceleration state of the platform is computed and subsequently, the actuator speeds and accelerations are determined. The analysis is further extended to determine the forces in the actuators.

Algorithm Implementation

The objective of the algorithm is to provide coefficients for the command equations (4.37)-(4.39)developed two sections ago. Another objective of the algorithms is to provide feedback in terms of the leg velocities and accelerations, and leg forces, to enable the designer to check the viability of the commands; i.e., by getting the feedback it can be determined whether the hardware would support and be able to match the commands (generally, a velocity command with limits on velocity and acceleration).

Initially, an algorithm (Movplat1c.m, Appendix B) was implemented, which when given a desired pose, velocity, and acceleration, would compute the position, velocity, and acceleration of the connectors using the inverse analysis (reverse66.m, Appendix B) and subsequently would generate the coefficients for the command equations to meet the position, velocity, and acceleration conditions.

For this case, the platform design parameters were taken as follows:

SIDE = 30 inches
bsacle = 0.8
tsidescale = 1.0
tscale = 0.8
hscale = 1.0

and the desired motion was a pure rotational motion defined by the motion of the center point of the platform about the z-axis (vertical). The rotation was described by the

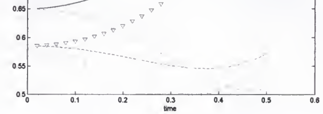


Figure 4.27: Leg Displacements for the 6-6 platform, executing a pure rotation.

following rotation matrix:

$$R = \begin{bmatrix} \cos(5\Delta t) & -\sin(5\Delta t) & 0 \\ \sin(5\Delta t) & \cos(5\Delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The algorithm computed the six leg displacements, velocities, and accelerations. Based on this information, the coefficients were calculated for each leg from Equations (4.40), (4.41), (4.43), (4.50), (4.51), and (4.52). Figures 4.27 through 4.29 show the leg displacements, leg velocities, and leg accelerations of the manipulator.

This analysis was further extended to determine the forces in the platform actuators. Using the concepts of dynamic analysis of rigid bodies using Screw Theory facilitated this. The equilibrium equation for the platform can be written as

$$[J] f_{legs} = W \tag{4.53}$$

where $[J]$ is the Jacobian composed of the screw coordinates of the platform actuators, f_{legs} are the forces in the legs and W is the wrench on the platform.

The wrench on a rigid body can be determined as

$$W = I A_R + [V \quad IV] \tag{4.54}$$

where I is a 6×6 inertial matrix and is composed as follows

$$I_{6\times6} = \begin{bmatrix} -MR & MI_d \\ I_{3\times3} & MR \end{bmatrix}. \tag{4.55}$$

MR is a coordinate transformation for different reference frames, and for this particular case, is a 3×3 matrix of zeroes. MI_d is the mass matrix times the identity matrix which produces a diagonal mass matrix. And finally, the $I_{3\times3}$ is the inertial matrix given as

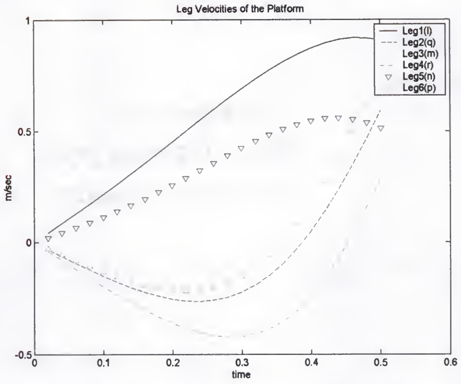


Figure 4.28: Leg velocities for the 6-6 platform executing a pure rotation.

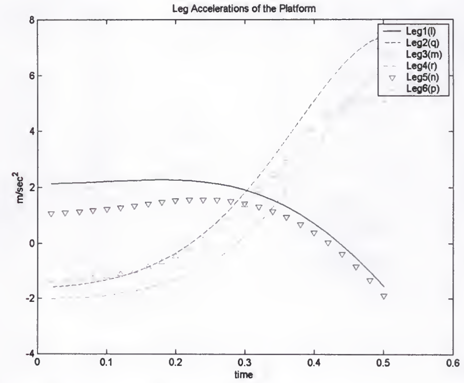


Figure 4.29: Leg accelerations for the 6-6 platform executing a pure rotation.

$$I_{3\times3} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{xz} & I_{yz} & I_{zz} \end{bmatrix}. \tag{4.56}$$

Further, A_R is the reduced acceleration state and V is the velocity state of the top platform and are determined as

$$A_R = \begin{bmatrix} \vec{\alpha} \\ \vec{a}_0 - \vec{\omega} \times \vec{\upsilon}_0 \end{bmatrix} \tag{4.57}$$

$$V = \begin{bmatrix} \vec{\omega} \\ \vec{\upsilon}_0 \end{bmatrix}. \tag{4.58}$$

And finally, the term in the square brackets, $[V \quad IV]$, is the Lie product of two screws and is determined as shown in Equation 2.55. Once all the components are known, the forces in the actuators are calculated from (4.53) as

$$f_{legs} = [J]^{-1} W. \tag{4.59}$$

This procedure was implemented using data for a platform being developed at CIMAR as a force compensating manipulator device. The mass and inertial matrices were formed from this data. The mass matrix was found to be

$$MI_d = \begin{bmatrix} 79.5928 & 0 & 0 \\ 0 & 79.5928 & 0 \\ 0 & 0 & 79.5928 \end{bmatrix} \text{kg}. \tag{4.60}$$

and the inertial matrix was calculated as

$$I_{3\times3} = \begin{bmatrix} 3.39146 & 0 & 0 \\ 0 & 3.39146 & 0 \\ 0 & 0 & 6.59053 \end{bmatrix} \text{kg.m}^2 \tag{4.61}$$

The function that calculated the inverse position, velocity, and acceleration for the previous case, was further developed to include the force analysis (reverse66f.m, Appendix B). This algorithm was tested for a variety of cases, such as constant velocity motion (translation), translation with constant acceleration, rotation with constant angular acceleration, etc. The results from two cases considered are presented:

- In this case, a constant acceleration motion was implemented as the platform moved in the vertical direction; results are shown in Figures 4.30 through 4.33 (Movplat2b.m, Appendix B).
- A rotation of the top platform about the center (z-axis) with no translation was implemented, with constant angular acceleration; results obtained from this simulation are presented in Figures 4.34 through 4.37 (Movplat2d.m, Appendix B).

Techniques were developed in this chapter for generating command screws that meet the desired position and velocity specifications for autonomous robots, serial manipulators, and parallel manipulators. The following chapter presents the case of a serial manipulator with a control system implementation.

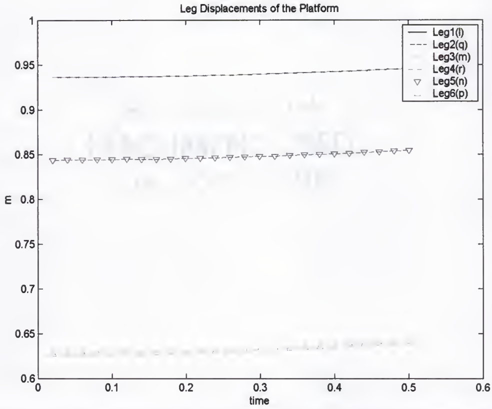


Figure 4.30: Leg displacements for a platform motion with translation and constant acceleration.

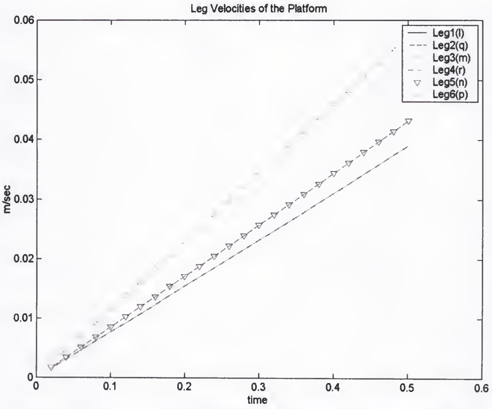


Figure 4.31: Leg velocities for a platform motion with translation and constant acceleration.

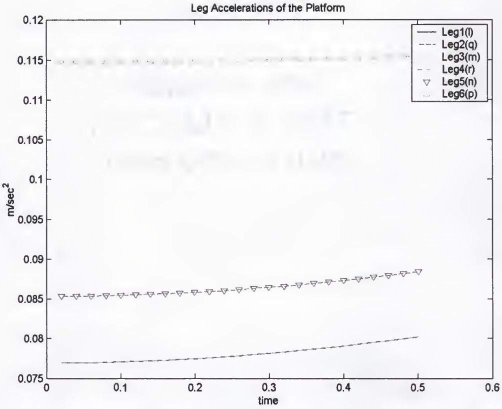


Figure 4.32: Leg accelerations for a platform motion with translation and constant acceleration.

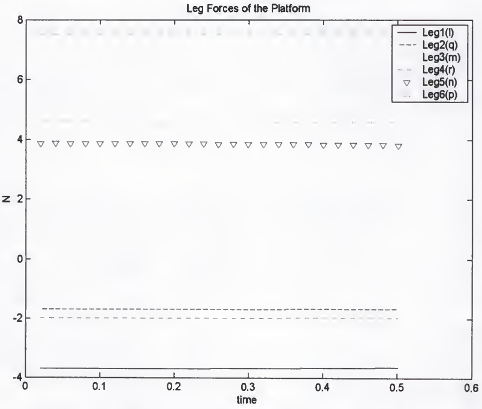


Figure 4.33: Leg forces for a platform motion with translation and constant acceleration.

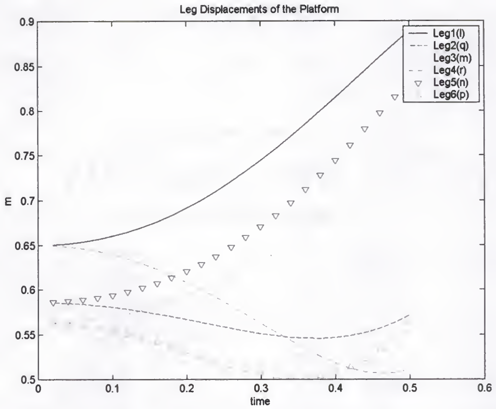


Figure 4.34: Leg displacements for a platform motion with rotation about the vertical axis, no translation, and constant angular acceleration.

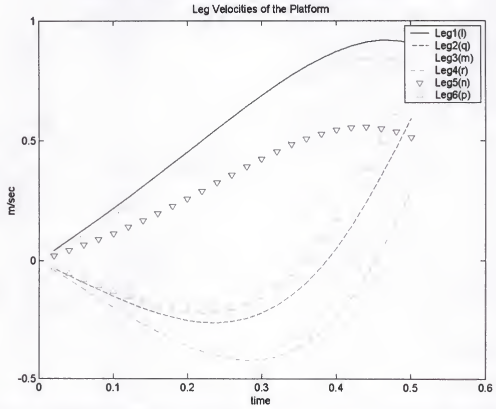


Figure 4.35: Leg velocities for a platform motion with rotation about the vertical axis, no translation, and constant angular acceleration.

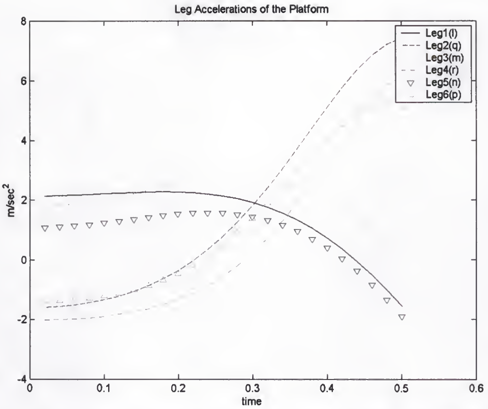


Figure 4.36: Leg accelerations for a platform motion with rotation about the vertical axis, no translation, and constant angular acceleration.

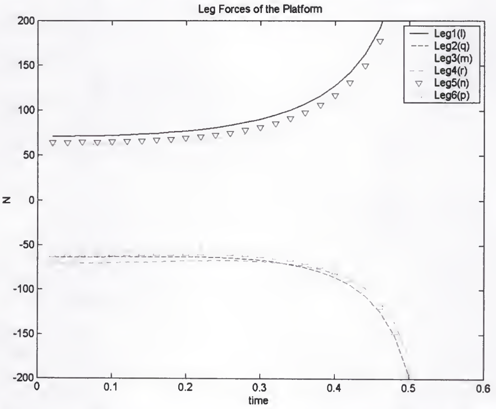


Figure 4.37: Leg forces for a platform motion with rotation about the vertical axis, no translation, and constant angular acceleration.

CHAPTER 5
PATH TRACKING AND CONTROL

This section discusses the control of a manipulator by developing a system model and applying the path tracking techniques developed in Chapter 4.

Manipulator Model

A manipulator model was developed for testing the path tracking algorithms developed and to study the system response. The model was based on a three link planar manipulator with all three joints being revolute. The link lengths were selected which defined the geometry of the manipulator as well as the motor torque requirements. It was decided to use electric motors as direct current (DC) motors lends themselves well to speed control applications. The torque requirements were calculated based on the mechanism geometry and performance desired. The mechanism parameters are (Figure 5.1):

l_1 = 30 cm, l_2 = 30 cm, and $l3$ = 15 cm, with a payload of 2.5 kgs.

Dynamic equations of the motor are used to determine the motor transfer function. The transfer function is then used to model the system parameters. The

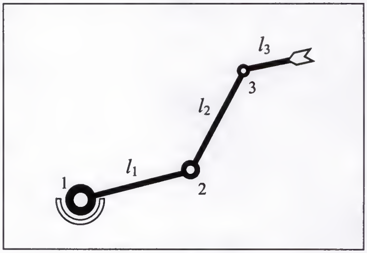


Figure 5.1: A 3R Planar Manipulator.

electrical equation of a motor is given as [Rel]:

$$V = L_a \frac{dI_a}{dt} + RI_a + K_E \omega \tag{5.1}$$

where

V = motor voltage

L_a = motor inductance

I_a = motor current

R = motor resistance

K_E = electrical constant, and

ω = motor velocity.

The magnetic field in the motor is constant and thus the current produces a proportional torque (T_g),

$$T_g = K_T I_a . \tag{5.2}$$

The opposing torque of the motor (T_m) is a function of the constant friction torque (T_f), and the viscous friction proportional to the velocity ($D\omega$), and can be written as

$$T_m = T_f + D\omega . \tag{5.3}$$

The relationship between the torques and velocity is the following:

$$T_g = (J_m + J_l)\frac{d\omega}{dt} + D\omega + T_f + T_l \tag{5.4}$$

where J_m and J_l are the moments of inertia of the motor and the load respectively, and T_l is the load opposing torque. Equation (5.4) is the dynamic equation of the motor, and along with Equations (5.1) and (5.2), it describes the relations between the electrical and the mechanical variables.

When a motor is used as a component in a system, the transfer function of the motor is required which can be determined from the equations described above. Assuming T_l and T_f equal to zero, since neither effect the motor transfer function, Equations (5.1), (5.2), and (5.4) can be written in the Laplace domain as

$$V(s) = (sL_a + R)I_a(s) + K_E\omega(s) \tag{5.5}$$

$$T_g(s) = K_T I_a(s) \tag{5.6}$$

$$T_g(s) = (J_m + J_l)s\omega(s) + D\omega(s)\,. \tag{5.7}$$

Representing the total moment of inertia as J, Equations (5.6) and (5.7) can be combined to derive an expression for the current which when substituted in Equation (5.5) yields

$$V(s) = \frac{1}{K_T}(sL_a + R)(sJ + D)\omega(s) + K_E\omega(s) \tag{5.8}$$

and the transfer function of the motor is

$$\begin{aligned} G_m(s) &= \frac{\omega(s)}{V(s)} \\ &= \frac{K_T}{(sL_a + R)(sJ + D) + K_E K_T}\,. \end{aligned} \tag{5.9}$$

The transfer function has two poles, which in all practical cases are negative and real. The transfer function can be written as

$$\begin{aligned} G_m(s) &= \frac{1/K_E}{(s\tau_1 + 1)(s\tau_2 + 1)} \\ &= \frac{K_T}{JL_a}\frac{1}{(s - p_1)(s - p_2)} \end{aligned} \tag{5.10}$$

where τ_1 and τ_2 are the time constants and p_1, p_2 are the poles of the transfer function. The time constants are related to the poles by

$$\tau_1 = -1/p_1 \text{ and } \tau_2 = -1/p_2 . \tag{5.11}$$

Considering the case where the damping factor D is negligible (=0) and the inductance L_a is small, the transfer function becomes

$$G_m(s) \quad = \quad \frac{K_T}{s^2 L_a J + sRJ + K_E K_T} \tag{5.12}$$

and the poles are the roots of the characteristic equation

$$s^2 L_a J + sRJ + K_E K_T = 0 . \tag{5.13}$$

Using the approximation for L_a being small, the poles are found to be

$$p_1 = \frac{-K_E K_T}{RJ} \text{ and } p_2 = \frac{-R}{L_a} \tag{5.14}$$

and from Equation (5.11) the time constants are

$$\tau_m = \frac{RJ}{K_E K_T} \text{ and } \tau_e = \frac{L_a}{R} \tag{5.15}$$

where τ_m is the *mechanical time constant* and τ_e is the *electrical time constant.* The relationship between the poles and the constants is valid only when the mechanical time constant is an order of magnitude larger than the electrical time constant, which is true from most DC brushless servo motors.

Motor Specifications

The torque requirements for the motors were developed based on the manipulator size and desired performance. Based on the requirements, DC brushless servomotors were the motors of choice; and specifications are shown in Table 5.1.

Table 5.1 Manipulator specifications.

Link	1	2	3
Link Length	30 cm	30 cm	15 cm
Torque Requirement	4.97 N.m	1.12 N.m	0.26 N.m
Motor Selected	Electro-Craft F 4050	Electro-Craft Y 2012-1	Electro-Craft LD 2003
Torque (cont./max.)	5.2/13.6 N.m	1.4/3.8 N.m	.34/1.02 N.m

System Model

For the purpose of this analysis, the motors are modeled as a first order system and step-response modeling is carried out. The basis for this assumption is the fact that overdamped second order and higher order systems can often be approximated by a first order system with dead time [Bol88]. The step-response indicates the form of the process and the coefficients in the process model can be determined. The input output relationship (transfer function) of a first order system in the Laplace domain is given by

$$\frac{y(s)}{x(s)}=\frac{1}{\tau s+1} \tag{5.16}$$

where τ is the time constant. Applying a unit step function $1/s$ as input in the equation above, we get

$$y(s)=\frac{1}{s}\frac{1}{\tau s+1}$$

and rearranging

$$y(s)=\frac{1}{s}-\frac{\tau}{\tau s+1} \tag{5.17}$$

and subsequently taking the inverse Laplace transform yields

$$y(s)=1-e^{-\frac{t}{\tau}}. \tag{5.18}$$

It should be noted from Equation (5.18) that the smaller the time constant τ, the faster the system response.

For implementing the system model, a discrete process model was used. The discrete transfer function is based on the use of difference equations. The backward shift operator B is used, which is defined as

$$By_n \equiv y_{n-1},\ B^2 y_n \equiv y_{n-2},\ \dots,\ B^j y_n \equiv y_{n-j}. \tag{5.19}$$

The discrete transfer function for a first order system with a time constant is given by

$$\frac{y_n}{x_n} = \frac{K(1-e^{-T/\tau})B}{1-e^{-T/\tau}B}, \tag{5.20}$$

and the discrete transfer function for a delay process is given by

$$\frac{y_n}{x_n} = KB^d, \tag{5.21}$$

where

$T =$ sample time

$\tau =$ time constant of the motor

$B =$ backward shift operator, and

$K =$ motor gain

$D =$ delay, and

$d = D/T =$ an integer.

The transfer function for a first order process with a delay then becomes

$$\frac{y_n}{x_n} = \frac{K(1-e^{-T/\tau})B^{d+1}}{1-e^{-T/\tau}B} \tag{5.22}$$

and is shown in Figure 5.2.

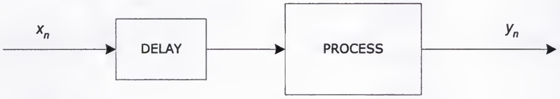


Figure 5.2: A process with delay.

Controller Development

Once the process (motor) model has been developed, the design of an appropriate controller was the next step. The objective being to design a controller that will allow a response that meets the transient and steady state requirements of the closed loop system. The most widely used controller in industrial control systems, the proportional-integral-derivative (PID) controller is a combination of the three basic control actions and provides the advantages of all the three individual control actions. The equation of a controller with PID action is given by

$$y(t) = K_p e(t) + \frac{K_p}{T_i}\int_0^t e(t)dt + K_p T_d \frac{de(t)}{dt} \tag{5.23}$$

where e is the error between the input and the output, K_p is the proportional gain, T_i is the integral time, and T_d is the derivative time. The transfer function of a PID controller is

$$\frac{y(s)}{e(s)} = K_p\left(1 + \frac{1}{T_i s} + T_d s\right) \tag{5.24}$$

or alternatively it can be written as

$$\frac{y(s)}{e(s)} = K_p + \frac{K_i}{s} + K_d s \tag{5.25}$$

where K_i is the integral gain and K_d is the derivative gain. In this case, when a controller is tuned, the gains are the adjustable parameters.

Once the controller type was selected, controller tuning for the process was done. Controller tuning is a method of selecting the controller parameters to meet the desired performance specifications. The method employed for tuning was the Ziegler and Nichols rules for tuning PID controllers. The Ziegler and Nichols rules [Oga97] are based on experimental step responses or the value of K_p that results in marginal stability when only the proportional control action is used.

The second method of Ziegler and Nichols rules was used for determining the controller parameters. In this method, the integral time T_i is set to infinity and the derivative time T_d is set to zero. Using the proportional control action only, the proportional gain is increased from 0 to a critical value K_{cr} where the output first exhibits sustained oscillations. The critical gain K_{cr} and the corresponding period P_{cr} are experimentally determined. The critical period is the time period for one complete oscillation of the output with critical proportional gain.

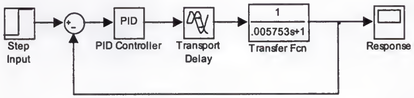


Figure 5.3: A closed loop system for controller tuning.

The tuning system was set up using the Simulink® software by modeling each motor in a feedback loop with a controller. The setup for each motor resembled the block diagram shown in Figure 5.3. The time constant, τ, for each motor was calculated using the data in Table 5.2 and was found to be

Table 5.2: Parameter values for determining the mechanical time constant of motors.

Motor	1 (F 4050)	2 (Y 2012-1)	3 (LD 2003)
R (Ohm)	.54	1.3	3.0
J (kg-m^2)	.0022	.000032	.000013
K_E (V/kRPM)	66	29	16
K_T (Nm/A)	.54	.24	.15

$$\tau_1 = 0.005753 \text{ sec.}$$

$$\tau_2 = 0.00063 \text{ sec.}$$

$$\tau_3 = 0.00170 \text{ sec.}$$

The integral and derivative gain values were set to zero and the proportional gain was varied until the response exhibited sustained oscillations. This gain was further tweaked to within a reasonable precision. Figure 5.4(a,b) illustrates this process. Once the critical gain was obtained, the critical time period was determined, and subsequently, using the Ziegler Nichols rules, the proportional gain was obtained as

$$K_p = 0.6\, K_{cr}, \tag{5.26}$$

following which, the integral time T_i, and the derivative time T_d was calculated

$$T_i = 0.5\, P_{cr} \tag{5.27}$$

$$T_d = 0.125\, P_{cr} \tag{5.28}$$

which finally yielded the gains as

$$K_i = \frac{K_p}{.5\, P_{cr}} \text{ and} \tag{5.29}$$

$$K_d = 0.125\, P_{cr}\, K_p. \tag{5.30}$$

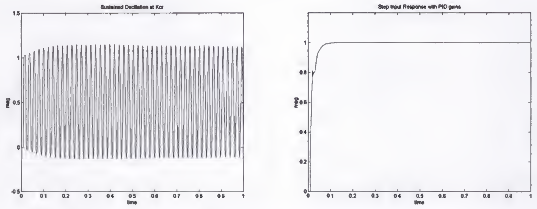


Figure 5.4: (a) Sustained oscillations at critical proportional gain (Motor 1). (b) Response to a step input with the PID controller (Motor 1).

It is important to note that the values obtained by this method are essentially an educated guess which provides a starting point for fine tuning the system, and hence the systems were fine tuned to obtain a smooth step response. It is, of course, also possible to add other advanced techniques such as feedforward control, disturbance rejection, adaptive control, or a combination thereof, for designing an optimal controller, but for the purpose of illustrating the path tracking in this case, PID control was deemed sufficient.

Controller Implementation

A discrete PID controller was implemented to control the process described in Equation (5.22). The controller is of the form

$$m_n = m_{n-1} + k_0\, er_n + k_1\, er_{n-1} + k_2\, er_{n-2} \tag{5.31}$$

where

$m_n =$ controller output

$er =$ (command – actual) error: controller input

$n-i =$ values at i time-steps back, and

$k_0,\ k_1, k_2$, are constants computed as

$$k_0 = K_p + K_i T + \frac{K_d}{T} \tag{5.32}$$

$$k_1 = -K_p - \frac{2K_d}{T} \tag{5.33}$$

$$k_2 = \frac{K_d}{T} \tag{5.34}$$

where T is the sample period.

Once the process transfer function and the controller transfer function were determined, the control loop was setup as shown in Figure 5.5. The discrete controller algorithm was implemented in MATLAB and the results were tested to correspond to the continuous time controller used in Simulink to model the step response. Once the results were verified, the control algorithm was amalgamated with the path-tracking algorithm for a 3R Manipulator.

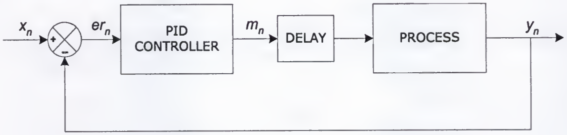


Figure 5.5: Closed loop PID controller for a process with delay.

The discrete path-tracking controller uses the following equation, based on Equation (5.22), to generate its output that is dependent on the previous value of the output and the control effort generated by the controller. Using Equation (5.22) and cross multiplying, we obtain

$$y_n\left(1-e^{-T/\tau}B\right) = K(1-e^{-T/\tau})B^{d+1}x_n \tag{5.35}$$

$$y_n - e^{-T/\tau}B\,y_n = K(1-e^{-T/\tau})B^{d+1}x_n \tag{5.36}$$

$$y_n = e^{-T/\tau}B\,y_n + K(1-e^{-T/\tau})B^{d+1}x_n. \tag{5.37}$$

Finally, applying the backward shift operator

$$y_n = e^{-T/\tau}\,y_{n-1} + K(1-e^{-T/\tau})x_{n-1-d}. \tag{5.38}$$

Writing the equation in terms of control variables we obtain,

$$\omega_{a(n)} = e^{-T/\tau}\,\omega_{a(n-1)} + K\left(1-e^{-T/\tau}\right)m_{n-1-d} \tag{5.39}$$

where

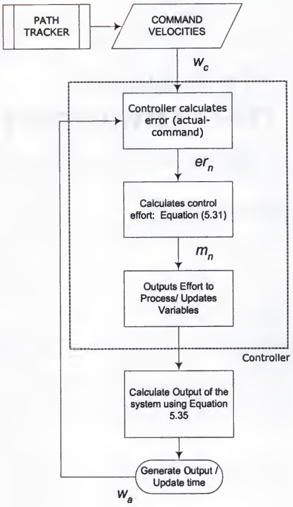
ω_a = system output (actual angular velocity) at time step n

$\omega_{a(n-1)}$ = actual angular velocity at previous time step

m_{n-1-d} = control effort at previous time step or earlier depending on the delay.

The controller was implemented for a 3R-manipulator path tracking using the methodology developed in Chapter 4. The flow chart for the algorithm is shown in Figure 5.6a and a block diagram of the process is shown in Figure 5.6b. The algorithm essentially represents an open loop tracking control of the manipulator. The controller loop operates at a higher frequency than the tracker command frequency. The option to adjust the controller loop frequency is a feature in the program. Another variable is the delay time D (d in Equation (5.39)), which can be fluctuated to study the behavior and response of the system and evaluate its performance.

Initially the program (main54cPID1c.m, Appendix B) was tested with a delay of 3 milliseconds (ms). To arrive at a reasonable estimate for a delay time, the following idea was used: a first order system, in response to a step input, reaches 99.3% of the final value in 5 time constants. Using this number and the motor with the highest time constant, and the assumption that the delay is approximately .5-5% of 5 time constants, the delay was found to be 1.5ms. For a conservative estimate, the delay used was doubled to account for any other delays in the computer control of the manipulator. Figures 5.7 and 5.8 illustrate the actual and command angular velocities for each motor and the errors between the two. This test was performed with a delay of 3ms.


PATH TRACKER
COMMAND VELOCITIES
w_c
Controller calculates error (actual-command)
er_n
Calculates control effort: Equation (5.31)
m_n
Outputs Effort to Process/ Updates Variables
Controller
Calculate Output of the system using Equation 5.35
Generate Output / Update time
w_a

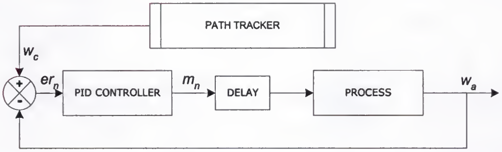


Figure 5.6: (a) Flowchart illustrating the algorithm for path tracking and control.
(b) Block diagram of the process.

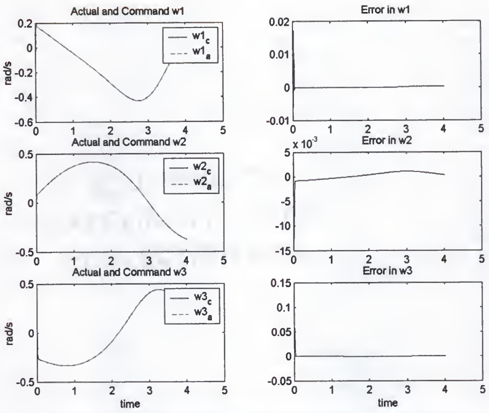


Figure 5.7: Actual and command angular velocities and errors

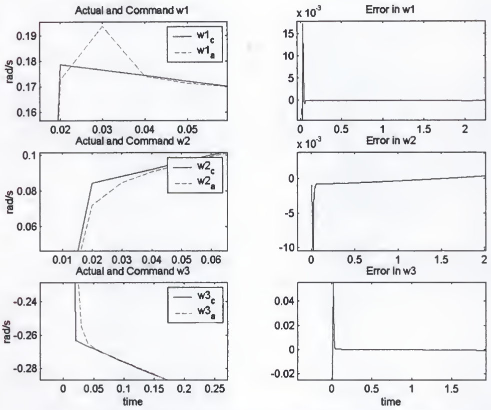


Figure 5.8: Magnified view of the command velocities and errors.

To determine the accuracy of the end effector of the manipulator, the errors in the end effector position were calculated. The error was determined by obtaining the difference between the command and actual positions (using the forward analysis) in both the x and y coordinates. Figure 5.9 illustrates these errors. Furthermore, the statistical error analysis yielded the following results:

Table 5.3: Errors in the end effector position.

Error	Mean (cm.)	Standard Deviation (cm.)
x-coordinate of the end effector	0.03786	0.05765
y-coordiante of the end effector	0.02666	0.06854

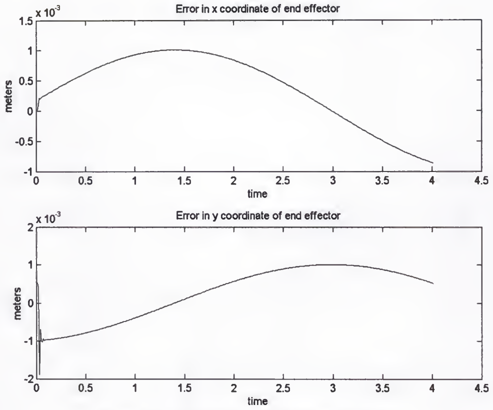


Figure 5.9: Errors in the end effector position.

On subsequent tests, the delay was increased step by step until the system became unstable and exhibited oscillatory response. The onset of oscillation was at 50ms and at higher delay times the system did not converge. Figure 5.10 shows the command and actual velocities at the 50ms-delay level. The onset of oscillation is apparent.

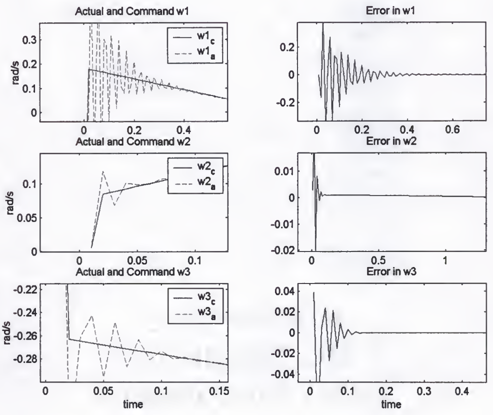


Figure 5.10: Actual and command angular velocities and errors with the Delay set at 50 milliseconds.

It is possible to retune the system to where it can handle the higher delay. The penalty of doing so is the higher rise time or sluggish response of the system. Experimentation was done for this purpose and the results showed that for systems tuned to a 100ms delay, the rise time was approximately 2.3 times that of a system tuned for a 10ms delay. A sample response test performed on motor 2 is shown in Figure 5.10a and Figure 5.10b.

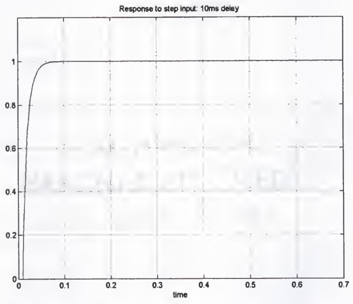


Figure 5.10(a): Step response for a system with 10ms delay.

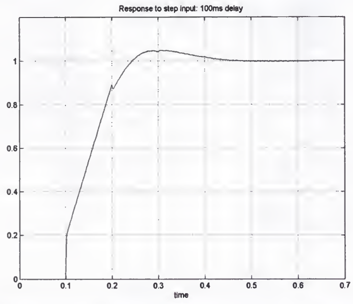


Figure 5.10(b): Step response for a system with 100ms delay.

The results of the simulation indicate that the controller will perform well within a range of delay times. If the delay times exceeds a threshold, the controller can be retuned although this would result in a sluggish response from the manipulator. It is anticipated that this method will produce similar results when applied to an experimental system.

CHAPTER 6
CONCLUSIONS AND FUTURE WORK

Conclusions

The Theory of Screws provides an elegant and useful method for analysis of mechanisms. In this research, the concepts of Screw Theory have been employed to develop path tracking methodologies for autonomous robots, serial robots, and parallel manipulators.

The method for autonomous robots and serial manipulators involves generating a command screw based on the constraints specified. The constraints can be at the position or the velocity level. For position level constraints, a command screw is generated based on the velocity screw. For velocity level constraints, a command screw is generated based on the combined effects of the velocity and acceleration screw. For autonomous robots, the accuracy of the generated command screw is verified by choosing a circular path and plotting the poles of the command screw. For serial robots, commands are generated using the reverse analysis, and the generated commands are compared to either the desired velocity state or the desired velocity and acceleration state depending on the constraints.

Tracking for parallel manipulators is accomplished by introducing a polynomial that satisfies conditions for each time step. The conditions were assumed to be the position, velocity, and acceleration at each time step. A fifth order polynomial is selected that has six constants which satisfy the position, velocity, and acceleration conditions at the beginning and end of each time step. The methodology involves the reverse analysis to generate the displacements, velocities, and accelerations, of the platform connectors based on the desired motion. Using this information the constants of the polynomial are determined which is used as the command for position, velocity, and acceleration.

The analysis for the parallel platforms was further extended to include the inverse dynamics to determine the forces in the legs. This was considered a beneficial tool for implementing a tracking and control strategy for a parallel manipulator as the output is an indicator of the hardware performance requirements of the system.

Path tracking with control was implemented on a 3R serial manipulator. This was done to study the performance of the path tracking algorithm with the dynamics of the system. A serial manipulator was modeled, which used electrical motors for actuation. The motors were modeled as a first order system with a delay. A PID controller was developed for motor control and was tuned for each motor. The results indicated that the controller followed the desired tracker commands within a band for the delay.

In summary, a methodology has been established that employs the Theory of Screws. It is by no means comprehensive but lays the foundation for Screw Theory based path tracking and control for general categories of robots and manipulators.

Future Work

The method developed for the autonomous robot does not consider vehicular constraints. The method can be extended to address constraints that are particular to an autonomous vehicle, such as vehicle with front wheel steering or vehicles with all wheel steering.

The methods do not consider constraints on velocities or accelerations. These limits can be incorporated into the algorithm when a specific application is known and the system limitations are known. The algorithm can then generate commands based on the best achievable value.

Another interesting prospect is the application of the parallel platform methodology to the spatial jack being developed at CIMAR. It is expected that this method will provide the basis for the tracking control of the platform. It is also expected that by being implemented on the manipulator, the algorithm will evolve to a better and more useful tool.

APPENDIX A
INITIAL PATH TRACKING APPROACHES

Initial path tracking approaches are presented here, the results of which led to the development of path tracking techniques discussed in the main text.

Considering the Pose and Velocity Information

In this case, the velocity state of the rigid body B is specified, in addition to the pose (position and orientation) information that is available. This is illustrated in Figure A.1.

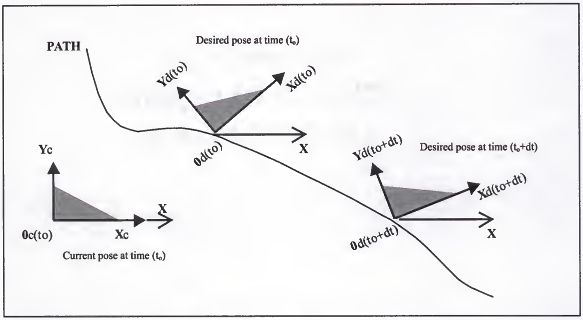

PATH
Desired pose at time (t_o)
Yd(to)
Xd(to)
X
Od(to)
Yc
X
Oc(to)
Xc
Current pose at time (t_o)
Yd(to+dt)
Desired pose at time (t_o+dt)
Xd(to+dt)
X
Od(to+dt)

Figure 4.1: A rigid body following a path with a prescribed pose and velocity.

In the most general case, the problem can be stated as: given the current pose of a rigid body B at time t_o, together with its velocity state at the same time t_o, and the desired timed path of the body, together with the desired time dependent specifications of the velocity state, develop a tracking technique so that at time $t_o+\Delta t$, the desired and the current poses and velocity states coincide, respectively.

The problem of computing rotations about arbitrary axes is quite complicated. At the outset, it was attempted to solve this problem with a simplified analysis involving rotations around a fixed axis combined with translations. This assumption is justified as to this point, as the initial analysis cases are two-dimensional. Once a technique is devised, it will be attempted to devise methods to include arbitrary spatial rotations.

Rotation around a fixed axis

In this case, the problem becomes as shown in Figure A.2. Given the current position θ_c at time t_o and the current velocity ω_c at time t_o, it is required that time $t_o+\Delta t$, both the desired and current position and the desired and current velocity state coincide.

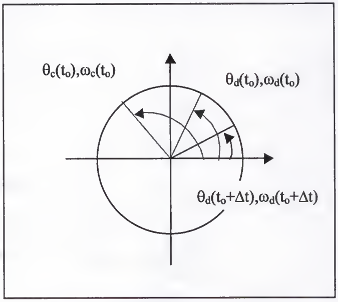


Figure A.2: Rotation about a fixed axis.

Assuming constant angular acceleration α, and integrating the angular speed equation $\omega = d\theta / dt$, can be written in difference form as:

$$\omega_c(t_0+\Delta t) = \omega_c(t_0) + \alpha\Delta t \qquad \text{(A.1)}$$
$$= \omega_d(t_0+\Delta t)$$

and integrating again,

$$\theta_c(t_0+\Delta t) = \theta_c(t_0) + \omega_c(t_0)\Delta t + \tfrac{1}{2}\alpha\Delta t^2 \qquad \text{(A.2)}$$

In general, it will be impossible to simultaneously solve (A.1) and (A.2) with only one variable, α, available. To overcome this problem, the following concept is introduced. Two acceleration states will be used, i.e., α_1 and α_2, such that

$$\alpha(t) = \begin{Bmatrix} \alpha_1 & t_0 < t < (t_0 + \frac{\Delta t}{2}) \\ \alpha_2 & (t_0 + \frac{\Delta t}{2}) < t < (t_0 + \Delta t) \end{Bmatrix} \qquad \text{(A.3)}$$

and is illustrated in Figure A.3. Two equations with two unknowns are now solvable. Integrating the equations as done previously, with a different time interval,

$$\omega_c(t_0+\tfrac{\Delta t}{2}) = \omega_c(t_0) + \alpha_1\frac{\Delta t}{2} \quad (A.4)$$

$$\omega_c(t_0+\Delta t) = \omega_c(t_0+\tfrac{\Delta t}{2}) + \alpha_2\frac{\Delta t}{2}$$

substituting, the former into the latter equation results in:

$$\omega_c(t_0+\Delta t) = \omega_c(t_0) + \frac{\alpha_1+\alpha_2}{2}\Delta t$$

and finally,

$$\omega_c(t_0) + \frac{\alpha_1+\alpha_2}{2}\Delta t = \omega_d(t_0+\Delta t) \qquad (A.5)$$

Similarly, for the orientation we have:

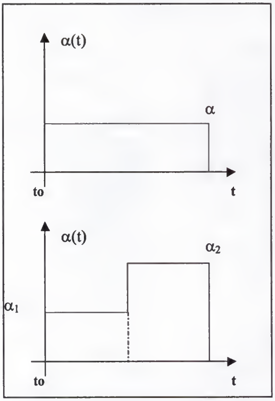


Figure A.3: Acceleration profile.

$$\theta_c(t_0+\tfrac{\Delta t}{2}) = \theta_c(t_0) + \omega_c(t_0)\frac{\Delta t}{2} + \frac{1}{2}\alpha_1\left(\frac{\Delta t}{2}\right)^2$$

$$\theta_c(t_0+\Delta t) = \theta_c(t_0+\tfrac{\Delta t}{2}) + \omega_c(t_0+\tfrac{\Delta t}{2})\frac{\Delta t}{2} + \frac{1}{2}\alpha_2\left(\frac{\Delta t}{2}\right)^2 \qquad (A.6)$$

and as done previously, substituting the former into the latter:

$$\theta_c(t_0+\Delta t) = \theta_c(t_0) + \omega_c(t_0)\frac{\Delta t}{2} + \frac{1}{2}\alpha_1\left(\frac{\Delta t}{2}\right)^2 + \omega_c(t_0+\tfrac{\Delta t}{2})\frac{\Delta t}{2} + \frac{1}{2}\alpha_2\left(\frac{\Delta t}{2}\right)^2$$

substituting the value for $\omega_c(t_0+\tfrac{\Delta t}{2})$ from Equation A.5

$$\theta_c(t_0+\Delta t) = \theta_c(t_0) + \omega_c(t_0)\frac{\Delta t}{2} + \frac{1}{2}\alpha_1\left(\frac{\Delta t}{2}\right)^2 + \left[\omega_c(t_0) + \alpha_1\frac{\Delta t}{2}\right]\frac{\Delta t}{2} + \frac{1}{2}\alpha_2\left(\frac{\Delta t}{2}\right)^2$$

$$\theta_c(t_0+\Delta t) = \theta_c(t_0) + \omega_c(t_0)\Delta t + \frac{3}{2}\alpha_1\left(\frac{\Delta t}{2}\right)^2 + \frac{1}{2}\alpha_2\left(\frac{\Delta t}{2}\right)^2$$

finally results in

$$\theta_d(t_0 + \Delta t) = \theta_c(t_0) + \omega_c(t_0)\Delta t + \frac{3}{2}\alpha_1\left(\frac{\Delta t}{2}\right)^2 + \frac{1}{2}\alpha_2\left(\frac{\Delta t}{2}\right)^2 \tag{A.7}$$

Equations (A.5) and (A.7) lead to the following system of linear equations:

$$\left.\begin{aligned} \alpha_1 + \alpha_2 &= \frac{2}{\Delta t}\left[\omega_d(t_0 + \Delta t) - \omega_c(t_0)\right] \\ 3\alpha_1 + \alpha_2 &= \frac{8}{\Delta t^2}\left[\theta_d(t_0 + \Delta t) - \theta_c(t_0) - \omega_c(t_0)\Delta t\right] \end{aligned}\right\} \tag{A.8}$$

Since the coefficient matrix of the above system is non-singular,

$$\begin{vmatrix} 1 & 1 \\ 3 & 1 \end{vmatrix} = 1 - 3 = -2 \neq 0,$$

the system always has a solution. The solution is unique assuming the time interval is halved; solutions can be obtained for other fractions of the time interval.

Translation in Space

The problem statement in this case is, given the current position vector $\mathbf{r}_c$ and velocity vector $\mathbf{v}_c$ at time t_o, as well as the desired position vector $\mathbf{r}_d$ and the velocity vector $\mathbf{v}_d$ find the motion states so that at time $t_o+\Delta t$, both the desired and current conditions coincide. This is shown in Figure A.4.

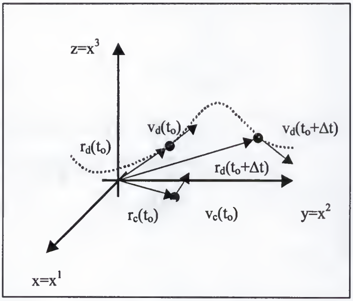


Figure A.4: Translation in space.

Since the solution is the same for each of the translation components, instead of employing the Cartesian x, y, z axes, the axes x^1, x^2, x^3 will be used. Furthermore, the problem is totally analogous to the rotation around the fixed axis problem, and thus the linear acceleration specification is given as:

$$a^i(t) = \begin{Bmatrix} a_1{}^i & t_0 < t < (t_0 + \frac{\Delta t}{2}) \\ a_2{}^i & (t_0 + \frac{\Delta t}{2}) < t < (t_0 + \Delta t) \end{Bmatrix}$$

And following the same procedure as in the prior section in the rotation case, the following equations are obtained:

$$v_c^i(t_0) + \frac{a_1^i + a_2^i}{2}\Delta t = v_d^i(t_0 + \Delta t) \tag{A.9}$$

$$r_d^i(t_0 + \Delta t) = r_c^i(t_0) + v_c^i(t_0)\Delta t + \frac{3}{2}a_1^i\left(\frac{\Delta t}{2}\right)^2 + \frac{1}{2}a_2^i\left(\frac{\Delta t}{2}\right)^2 \tag{A.10}$$

Thus, for any *i*=1,2,3, the linear system becomes,

$$\left.\begin{aligned} a_1^i + a_2^i &= \frac{2}{\Delta t}\left[v_d^i(t_0 + \Delta t) - v_c^i(t_0)\right] \\ 3a_1^i + a_2^i &= \frac{8}{\Delta t^2}\left[r_d^i(t_0 + \Delta t) - r_c^i(t_0) - v_c^i(t_0)\Delta t\right] \end{aligned}\right\} \tag{A.11}$$

<u>Amalgam of the Translation and Rotation Analyses</u>

The two previous analyses can be combined to provide a solution for the path tracking problem of a serial, non-redundant, planar manipulator. Combining the results of these analyses, the velocity and reduced acceleration states for the end-effector of the planar robot can be obtained. Considering the poses and the velocity states of the end-effector, the velocity and acceleration specifications[1] for the end-effector at time t=t$_0$ are:

[1] The index in the formulas goes from 0 to 3 because a 3-link manipulator is considered.

$$ {}^{0}\vec{V}^{3}(t_0) \;=\; \vec{V}_c(t_0) \;=\; \begin{bmatrix} {}^{0}\vec{\omega}_c{}^{3}(t_0) \\ {}^{0}\vec{v}_0{}^{3}(t_0) \end{bmatrix} \tag{A.12} $$

$$ {}^{0}\vec{A}_R{}^{3}(t_0) \;=\; \begin{bmatrix} {}^{0}\vec{\alpha}_1{}^{3}(t_0) \\ {}^{0}\vec{a}_{01}{}^{3}(t_0) - {}^{0}\vec{\omega}_c{}^{3}(t_0) \times {}^{0}\vec{v}_0{}^{3}(t_0) \end{bmatrix} \tag{A.13} $$

where ${}^{0}\vec{\alpha}_1{}^{3}(t_0)$ and ${}^{0}\vec{a}_{01}{}^{3}(t_0)$ are obtained from the analyses done in previous sections.

The joint velocities and accelerations can be obtained from the following set of equations [Ric99]:

$$ {}^{0}\vec{V}^{3}(t_0) \;=\; {}_{0}\omega_1(t_0)\,{}^{0}\$^{1}(t_0) + {}_{1}\omega_2(t_0)\,{}^{1}\$^{2}(t_0) + {}_{2}\omega_3(t_0)\,{}^{2}\$^{3}(t_0) \tag{A.14} $$

$$ {}^{0}\vec{A}_R{}^{3}(t_0) \;=\; \left\{ \begin{array}{l} {}_{0}\dot{\omega}_1(t_0)\,{}^{0}\$^{1}(t_0) + {}_{1}\dot{\omega}_2(t_0)\,{}^{1}\$^{2}(t_0) + {}_{2}\dot{\omega}_3(t_0)\,{}^{2}\$^{3}(t_0) + \\ \left[{}_{0}\omega_1(t_0)\,{}^{0}\$^{1}(t_0) \quad {}_{1}\omega_2(t_0)\,{}^{1}\$^{2}(t_0) + {}_{2}\omega_3(t_0)\,{}^{2}\$^{3}(t_0) \right] + \\ \left[{}_{1}\omega_2(t_0)\,{}^{1}\$^{2}(t_0) \quad {}_{2}\omega_3(t_0)\,{}^{2}\$^{3}(t_0) \right] \end{array} \right\} \tag{A.15} $$

where the quantities in [ ] brackets represent the Lie product of two screws.

For time t=$(t_0 + \frac{\Delta t}{2})$, the velocity and reduced acceleration expressions become:

$$ {}^{0}\vec{V}^{3}(t_0 + \tfrac{\Delta t}{2}) \;=\; \begin{bmatrix} {}^{0}\vec{\omega}_c{}^{3}(t_0) + {}^{0}\vec{\alpha}_1{}^{3}(t_0)\dfrac{\Delta t}{2} \\ {}^{0}\vec{v}_0{}^{3}(t_0) + {}^{0}\vec{a}_{01}{}^{3}(t_0)\dfrac{\Delta t}{2} \end{bmatrix} \tag{A.16} $$

$$ {}^{0}\vec{A}_R{}^{3}(t_0 + \tfrac{\Delta t}{2}) \;=\; \begin{bmatrix} {}^{0}\vec{\alpha}_2{}^{3}(t_0) \\ {}^{0}\vec{a}_{02}{}^{3}(t_0) - \left[{}^{0}\vec{\omega}_c{}^{3}(t_0) + {}^{0}\vec{\alpha}_1{}^{3}(t_0)\dfrac{\Delta t}{2} \right] \times \left[{}^{0}\vec{v}_0{}^{3}(t_0) + {}^{0}\vec{a}_{01}{}^{3}(t_0)\dfrac{\Delta t}{2} \right] \end{bmatrix} \tag{A.17} $$

The new joint velocities and accelerations are obtained from Equation (A.16) and Equation (A.17) evaluated at t=$(t_0 + \frac{\Delta t}{2})$ as follows:

$$ {}^{0}\vec{V}^{3}(t_0+\tfrac{\Delta t}{2}) = \left\{ \begin{array}{l} {}_{0}\omega_1(t_0+\tfrac{\Delta t}{2})\ {}^{0}\$^{1}(t_0+\tfrac{\Delta t}{2}) + {}_{1}\omega_2(t_0+\tfrac{\Delta t}{2})\ {}^{1}\$^{2}(t_0+\tfrac{\Delta t}{2}) \\ +\ {}_{2}\omega_3(t_0+\tfrac{\Delta t}{2})\ {}^{2}\$^{3}(t_0+\tfrac{\Delta t}{2}) \end{array} \right\} \tag{A.18} $$

$$ {}^{0}\vec{A}_R{}^{3}(t_0+\tfrac{\Delta t}{2}) = \left\{ \begin{array}{l} {}_{0}\dot{\omega}_1(t_0+\tfrac{\Delta t}{2})\ {}^{0}\$^{1}(t_0+\tfrac{\Delta t}{2}) + {}_{1}\dot{\omega}_2(t_0+\tfrac{\Delta t}{2})\ {}^{1}\$^{2}(t_0+\tfrac{\Delta t}{2}) \\ +\ {}_{2}\dot{\omega}_3(t_0+\tfrac{\Delta t}{2})\ {}^{2}\$^{3}(t_0+\tfrac{\Delta t}{2}) + \\ \left[{}_{0}\omega_1(t_0+\tfrac{\Delta t}{2})\ {}^{0}\$^{1}(t_0+\tfrac{\Delta t}{2}) \quad {}_{1}\omega_2(t_0+\tfrac{\Delta t}{2})\ {}^{1}\$^{2}(t_0+\tfrac{\Delta t}{2}) + {}_{2}\omega_3(t_0+\tfrac{\Delta t}{2})\ {}^{2}\$^{3}(t_0+\tfrac{\Delta t}{2})\right] \\ +\left[{}_{1}\omega_2(t_0+\tfrac{\Delta t}{2})\ {}^{1}\$^{2}(t_0+\tfrac{\Delta t}{2}) \quad {}_{2}\omega_3(t_0+\tfrac{\Delta t}{2})\ {}^{2}\$^{3}(t_0+\tfrac{\Delta t}{2})\right] \end{array} \right\} \tag{A.19} $$

It must be pointed out that during the motion from $t=t_0$ to $t=t_0+\Delta t$, there will be changes in the location of the screws which would yield a result that would not be exact, and as a consequence of which, there would be a small error between the desired and required poses and velocity states. However, it can be hypothesized that with the selection of a small interval Δt, the effect of this error can be minimized.

Path Tracking Assuming a Linearly Varying Acceleration Profile

Consider the initial and final acceleration after a time step Δt:

$$ a_0 = a(t) \text{ at } t_0 $$

$$ a_f = a(t) \text{ at } t_0 + \Delta t. $$

Now assuming that the variation is linear, it can be written that

$$ a(t) = a_0 + \frac{a_f - a_0}{\Delta t}(t - t_0) \tag{A.20} $$

$$ a(t) \approx a_0 + \frac{t}{\Delta t}(a_f - a_0) \tag{A.21} $$

then

$$ \upsilon(t) = a_0 t + \frac{t^2}{2\Delta t}(a_f - a_0) + C_1. \tag{A.22} $$

For t=0, $\upsilon(t) = \upsilon_0$, thus

$$ \upsilon(t) = \upsilon_0 + a_0 t + \frac{t^2}{2\Delta t}(a_f - a_0). \tag{A.23} $$

Integrating again,

$$s(t) = \upsilon_0 t + \frac{a_0 t^2}{2} + \frac{t^3}{6\Delta t}\left(a_f - a_0\right) + C_2 . \tag{A.24}$$

Again, for t=0, $s(t) = s_0$, thus

$$s(t) = s_0 + \upsilon_0 t + \frac{a_0 t^2}{2} + \frac{t^3}{6\Delta t}\left(a_f - a_0\right). \tag{A.25}$$

The values of a_f and a_0 must satisfy the conditions

$$s(t) = s_f \text{ at } t_0 + \Delta t \text{ and}$$

$$\upsilon(t) = \upsilon_f \text{ at } t_0 + \Delta t$$

therefore

$$\upsilon_f = \upsilon_0 + a_0 \Delta t + \frac{\Delta t^2}{2\Delta t}\left(a_f - a_0\right) \tag{A.26}$$

and

$$s(t) = s_0 + \upsilon_0 \Delta t + \frac{a_0 \Delta t^2}{2} + \frac{\Delta t^3}{6\Delta t}\left(a_f - a_0\right). \tag{A.27}$$

Rearranging the equations,

$$\upsilon_f - \upsilon_0 = \Delta t\left(a_0 + \frac{a_f}{2} - \frac{a_0}{2}\right) \tag{A.28}$$

$$s_f - s_0 = \upsilon_0 \Delta t + \Delta t^2\left(\frac{a_0}{2} + \frac{a_f}{6} - \frac{a_0}{6}\right) \tag{A.29}$$

or

$$\upsilon_f - \upsilon_0 = \frac{\Delta t}{2}\left(a_f + a_0\right) \tag{A.30}$$

$$s_f - s_0 - \upsilon_0 \Delta t = \Delta t^2\left(\frac{a_0}{3} + \frac{a_f}{6}\right) \tag{A.31}$$

and finally

$$\frac{2(\upsilon_f - \upsilon_0)}{\Delta t} = (a_f + a_0) \tag{A.32}$$

$$\frac{6(s_f - s_0 - \upsilon_0 \Delta t)}{\Delta t^2} = 2a_0 + a_f\,. \tag{A.33}$$

Solving for a_0 and a_f

$$a_0 = -\frac{2(\upsilon_f \Delta t - 3s_f + 3s_0 + 2\upsilon_0 \Delta t)}{\Delta t^2} \tag{A.34}$$

$$a_f = \frac{2(2\Delta t\, \upsilon_f - 3s_f + 3s_0 + \upsilon_0 \Delta t)}{\Delta t^2}\,. \tag{A.35}$$

APPENDIX B
SIMULATION SOURCE CODE

Source code for the simulations using MATLAB®.

```
% autonomous.m
% Implementing path tracking on an Autonomous robot
% or Vehicle.
% Using the Position and Orientation information ONLY
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
% (No velocity information)
%
% Previous: particle.m
% Next: autonomous1.m

D2R = pi/180.; % Degrees to Radians conversion factor
r = 0.5;
t0 = 0.0;
Cx=1.0;
Cy=0.0;

tf=6.20;        % 6.5
delta = 0.05 ;

% Initializing all the vector arrays
ta = [];
x0da=[];
y0da=[];
r0xa = [];
r0ya = [];
r01c=[1.51; -0.05; 0];
thetac=1.7;

% The Loop starts over here for the duration tf

while t0<tf

rd = [Cx + r*cos(t0+delta); Cy+r*sin(t0+delta); 0];
they=atan2(rd(2,1)-Cy,rd(1,1)-Cx);
if they<0
  thetad=they + 2.0*pi + pi/2;
else
   thetad = they+pi/2;
end;

% Calculations for computing the required velocities
% Calculating the angular velocities and accelerations

V0c=[ 0; 0; (thetad-thetac)/delta;(rd(1,1)-r01c(1,1))/delta;(rd(2,1)-
r01c(2,1))/delta;0];
w=[ 0 0 V0c(3,1)];
V0=[V0c(4,1) V0c(5,1) 0];
r0=r01c+(Cross(w,V0)/V0c(3,1)^2)';

%The location of the velocity pole is
r0x = r0(1,1);
r0y = r0(2,1);

ta = [ ta [ t0 + delta ]];
r0xa = [ r0xa [ r0x ]];
r0ya = [ r0ya [ r0y ]];
x0da = [ x0da [ rd(1,1) ]];
```

```
y0da = [ y0da [ rd(2,1) ]];

r01c=rd;
thetac=thetad;
t0 = t0 + delta;

end

figure;

plot(r0xa,r0ya,'o'), title ('Path Tracking (pose) : Screw Locations
"o"')
hold on
plot(x0da,y0da,'r+')
%axis([-2 2 -2 2])
legend ('pole locations','path');
axis equal;
zoom on;
grid on;
hold off
```

```
% autonomous1.m
% MAIN PROGRAM
% Implementing path tracking on an Autonomous robot
% or Vehicle.
% Using the Position and Orientation information ONLY
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
% (No velocity information)
%
% Previous: autonomous.m
% (the only new thing is different starting point )
% Next: autonomous2.m

D2R = pi/180.; % Degrees to Radians conversion factor
r = 0.5;
t0 = 0.0;
Cx=1.0;
Cy=0.5;

tf=1.9*pi;
delta = 0.05;

% Initializing all the vector arrays
ta = [];
x0da=[];
y0da=[];
r0xa = [];
r0ya = [];
thetada = [];
r01c=[1.50; 0.5; 0];
thetac=pi/2;

% The Loop starts over here for the duration tf

while t0<tf

rd = [Cx+r*cos(t0+delta); Cy+r*sin(t0+delta); 0];
they=atan2(rd(2,1)-Cy,rd(1,1)-Cx);
if they<0
  thetad=they + 2.0*pi + pi/2;
else
   thetad = they+pi/2;
end;

% Calculations for computing the required velocities
% Calculating the angular velocities

V0c=[ 0; 0; (thetad-thetac)/delta;(rd(1,1)-r01c(1,1))/delta;(rd(2,1)-
r01c(2,1))/delta;0];
w=[ 0 0 V0c(3,1)];
V0=[V0c(4,1) V0c(5,1) 0];
r0= r01c+(cross(w,V0)/V0c(3,1)^2)';

%The location of the velocity pole is
r0x = r0(1,1);
r0y = r0(2,1);
```

```
ta = [ ta [ t0 + delta ]];
r0xa = [ r0xa [ r0x ]];
r0ya = [ r0ya [ r0y ]];
x0da = [ x0da [ rd(1,1) ]];
y0da = [ y0da [ rd(2,1) ]];
thetada = [thetada [thetad]];

r01c=rd;
thetac=thetad;
t0 = t0 + delta;

end

figure;

plot(r0xa,r0ya,'o'), title ('Path tracking (pose) : Screw Locations
"o"')
hold on
plot(x0da,y0da,'r+')
legend ('pole location','path');
axis equal;
%axis([-2 2 -2 2])
grid on;
zoom on
hold off
```

```
% autonomous2a.m
% Implementing Path Tracking on an Autonomous Robot/Vehicle
% specifying the initial position and the final
% position and velocity states
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
% Circular Path / Draws the orientation as a box
%
% Previous: autonomous.m

D2R = pi/180.; % Degrees to Radians conversion factor
r = 0.5;
t0 = 0.0;
Cx=1.0;
Cy=0.5;

tf=6.1 ;  % original value 6.3
delta = 0.1;

% Initializing all the vector arrays
ta = [];
x0da=[];
y0da=[];
r0xa = [];
r0ya = [];
thetada = [];
angaa=[];
axa=[];
aya=[];
omegaa=[];
vxa=[];
vya=[];

r01c=[1.54; .45; 0];
thetac=pi/2+0.002;

% The Loop starts over here for the duration tf

while t0<tf
% path is specified as a circle at an offset (Cx, Cy) from the Origin
rd = [Cx+r*cos(t0+delta); Cy+r*sin(t0+delta); 0];
they=atan2(rd(2,1)-Cy,rd(1,1)-Cx);
if they<0
  thetad=they + 2.0*pi + pi/2;
else
   thetad = they+pi/2;
end;

vd = [-r*sin(t0+delta);r*cos(t0+delta);0];
omegad=1;

% Calculations for computing the required initial velocities
% Calculating the angular velocities and accelerations

V0c=[ 0; 0; 2*((thetad-thetac)/delta)-omegad;2*((rd(1,1)-
r01c(1,1))/delta)-vd(1,1);2*((rd(2,1)-r01c(2,1))/delta)-vd(2,1);0];
```

```
As=[0; 0; (2/delta)*(omegad-((1*(thetad-thetac))/delta)) ;
(2/delta)*(vd(1,1)-((1*(rd(1,1)-r01c(1,1)))/delta)) ;
(2/delta)*(vd(2,1)-((1*(rd(2,1)-r01c(2,1)))/delta)) ; 0]

w=[ 0 0 V0c(3,1)];
V0=[V0c(4,1) V0c(5,1) 0];
r0= r01c+(cross(w,V0)/V0c(3,1)^2)';

%The location of the initial velocity pole is
r0x = r0(1,1);
r0y = r0(2,1);

ta = [ ta [ t0 + delta ]];
r0xa = [ r0xa [ r0x ]];
r0ya = [ r0ya [ r0y ]];
x0da = [ x0da [ rd(1,1) ]];
y0da = [ y0da [ rd(2,1) ]];
thetada = [thetada [thetad]];
angaa = [angaa [As(3,1)]];
axa=[ axa [As(4,1)]];
aya=[ aya [As(5,1)]];
omegaa=[omegaa [V0c(3,1)]];
vxa=[vxa [V0c(4,1)]];
vya=[vya [V0c(5,1)]];

r01c=rd;
thetac=thetad;
t0 = t0 + delta;

end

figure;
plot(r0xa,r0ya,'o'), title ('Path tracking (Pose, Velocity) : Screw
Locations "o"')

hold on
for i = 1:length(x0da)
 DrawBox([x0da(i)  ; y0da(i)], thetada(i) , 0.02, 0.01);
end
plot(x0da,y0da,'g');
legend ('pole locations');
%axis([-2 2 -2 2])
axis equal;
zoom on;
hold off;
```

```
% autonomous3.m
% Autonomous Path Tracking
% specifying the initial position and the final
% position and velocity states: Initial velocity is NOT specified
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
% Implemented with a DrawBox function to show the change in
% orientation as the vehicle traverses the desired path (Hyperbolic
path)
% Previous: autonomous2.m

D2R = pi/180.; % Degrees to Radians conversion factor
r = 0.5;
t0 = -0.5;
Cx=1.0;
Cy=0.5;

tf=0.5;
delta = 0.05;

% Initializing all the vector arrays
ta = [];
x0da=[];
y0da=[];
r0xa = [];
r0ya = [];
thetada = [];
angaa=[];
axa=[];
aya=[];
omegaa=[];
vxa=[];
vya=[];

r01c=[4; 8; 0];
thetac=0.002;

% The Loop starts over here for the duration tf

while t0<tf

% defining the trjectory of the desired motion
% rd = [Cx+r*cos(t0+delta); Cy+r*sin(t0+delta); 0]; % circle
% rd = [ 6 ; 5*(t0+delta) ; 0 ]; % constant x, linear vel. y
rd = [ t0+delta ;  4*(t0+delta)^2 ; 0 ];  % parabola

thetad = 5.0*(t0+delta);

vd = [1 ; 8*(t0+delta) ; 0];
omegad=5.0;

% Calculations for computing the required initial velocities
% Calculating the angular velocities and accelerations

V0c=[ 0; 0; 2*((thetad-thetac)/delta)-omegad;2*((rd(1,1)-
r01c(1,1))/delta)-vd(1,1);2*((rd(2,1)-r01c(2,1))/delta)-vd(2,1);0];
As=[0; 0; 2*(thetad-thetac)/delta^2;2*(rd(1,1)-
r01c(1,1))/delta^2;2*(rd(2,1)-r01c(2,1))/delta^2; 0];
```

```
w=[ 0 0 V0c(3,1)];
V0=[V0c(4,1) V0c(5,1) 0];
r0= r01c+(cross(w,V0)/V0c(3,1)^2)';

%The location of the initial velocity pole is
r0x = r0(1,1);
r0y = r0(2,1);

ta = [ ta [ t0 + delta ]];
r0xa = [ r0xa [ r0x ]];
r0ya = [ r0ya [ r0y ]];
x0da = [ x0da [ rd(1,1) ]];
y0da = [ y0da [ rd(2,1) ]];
thetada = [thetada [thetad]];
angaa = [angaa [As(3,1)]];
axa=[ axa [As(4,1)]];
aya=[ aya [As(5,1)]];
omegaa=[omegaa [V0c(3,1)]];
vxa=[vxa [V0c(4,1)]];
vya=[vya [V0c(5,1)]];

r01c=rd;
thetac=thetad;
t0 = t0 + delta;

end

figure;
plot(r0xa,r0ya,'o'), title ('Path Tracking (Pose, Velocity) : Screw
locations "o"')
hold on
plot(x0da,y0da,'c')
legend ('pole locations','path');

for i = 1:length(x0da)
 DrawBox([x0da(i)  ; y0da(i)], thetada(i) , .05, 0.025);
end

axis([-2 2 -0.5 3.5])
zoom on
hold off
```

```
% main4.m
% An example for a 3R planar serial chain to study
% the proposed path tracking methodology
% using initial position and final position data
% Program requires the following functions:
% / Screw / Draw3R
%  Previous: main3.m
%  Next:  ??
% Last Update: June 10, 2001

D2R = pi/180.; % Degrees to Radians conversion factor
a_= [ 1 ; 1 ; 1];  % Constant data, link lengths

% r is a constant required by the end-effector path.
r = 0.2;

% The initial and final time, and the time step are given below
t0 = 0.0;
tf=6.34;
delta = 0.1;

%Current Position Data
rc = [2; 1 ; 0];
thetac = 0;
[set1c, set2c, flagc] = reverse_position_3R(a_, rc, thetac);
if flagc == 1
   error('The initial position is outside the manipulator workspace,
there is no point in completing the problem')
else
   % Initializing all the vector arrays
   ta = [];
   r0xa = [];
   r0ya =[];
   thetaa=[];
   theta1a=[];
   theta2a=[];
   theta3a=[];
   w01a=[];
   w12a=[];
   w23a=[];
   ew01a=[];
   ew12a=[];
   ew23a=[];
   %w01f=[];

   % The Loop starts over here for the duration tf
   while t0<tf

   %The desired motion and velocity of the end-effector is given below
   rd = [0.5+r*cos(t0+delta); r*sin(t0+delta); 0];
   thetad = 0;
   Vd=[0;0;0;-r*sin(t0+delta);r*cos(t0+delta); 0];

   % The following statement compute the reverse analysis of the
manipulator      % for the required position at time t0+delta.
   [set1, set2, flag] = reverse_position_3R(a_, rd, thetad);
   if flag == 1
```

```
      error('The required position is outside the manipulator
workspace')
   else
      [Jd,waste] = Screw(a_,set1);
      % wd are the join angular velocities computed from the desired
      % velocity  state
      wd=Jd\Vd;
      % w are the join angular velocities computed from our tracking
      % algorithm
      w=(set1-set1c)/delta;

      % Calculating Errors in Position, Orientation,
      % Linear and Angular velocity

      ta = [ ta [ t0 + delta ]];
      r0xa = [ r0xa [ rd(1,1) ]];
      r0ya = [ r0ya [ rd(2,1) ]];
      thetaa = [thetaa [thetad]];
      theta1a = [theta1a [set1(1,1)]];
      theta2a = [theta2a [set1(2,1)]];
      theta3a = [theta3a [set1(3,1)]];
      w01a=[w01a [ w(1,1)]];
      w12a=[w12a [ w(2,1)]];
      w23a=[w23a [ w(3,1)]];
      ew01a=[ew01a [w(1,1)-wd(1,1)]];
      ew12a=[ew12a [w(2,1)-wd(2,1)]];
      ew23a=[ew23a [w(3,1)-wd(3,1)]];

      % Updating for the next iteration
      t0 = t0 + delta;
      set1c=set1;

   end

end

   % some plotting
   figure;
   plot(r0xa,r0ya), title ('Tracking of 3R manip. using inv. analysis
(Pose)');

   Draw3R(a_,theta1a,theta2a,theta3a);
   % plot(ta, w01a, ta, w12a, ta, w23a),title('Joint Angular
Velocities');
   figure;
   plot(ta, w01a, ta, w01a+ew01a, ta, w12a, ta, w12a+ew12a, ta, w23a,
ta, w23a+ew23a), title('Actual and Desired Angular Velocities');

legend('omega01','omega01_d','omega12','omega12_d','omega23','omega23_d
');
   xlabel('time');
   figure;
   plot(ta, ew01a, '-', ta, ew12a, '--', ta, ew23a, ':'), title('Errors
in Joint Angular velocities');
   legend('error w01', 'error w12', 'error w23');
   xlabel('time');
end
```

```
% main54.m
% An example for a 3R planar serial chain to study
% the proposed path tracking methodology
% using initial position and final position data
% Program requires the following functions:
% / Screw / Draw3R
%  Previous: main4.m
%  Updated: June 15, 2001

D2R = pi/180.; % Degrees to Radians conversion factor
a_= [ 1 ; 1 ; 1];  % Constant data, link lengths

% r is a constant required by the end-effector path.
r = 0.2;

% The initial and final time, and the time step are given below
t0 = 0.0;
tf=6.34;
delta = 0.1 ;

%Current Position Data
rc = [0.72; 0.1 ; 0];
thetac = 0;
[set1c, set2c, flagc] = reverse_position_3R(a_, rc, thetac);
if flagc == 1
   error('The initial position is outside the manipulator workspace,
there is no point in completing the problem')
else
   % Initializing all the vector arrays
   ta = [];
   r0xa = [];
   r0ya =[];
   thetaa=[];
   theta1a=[]; theta2a=[];   theta3a=[];
   w01a=[];     w12a=[];      w23a=[];
   alpha01a=[]; alpha12a=[];  alpha23a=[];
   ew01a=[];    ew12a=[];     ew23a=[];
   eal01a=[];   eal12a=[];    eal23a=[];


   % The Loop starts over here for the duration tf
   while t0<tf

   %The desired motion and velocity of the end-effector is given below
   rd = [0.5+r*cos((t0+delta)); r*sin((t0+delta)); 0];
   thetad = 0.0;
   Vd=[0;0;0;-r*sin(t0+delta);r*cos(t0+delta); 0];
   As=[0;0;0;-r*cos(t0+delta);-r*sin(t0+delta); 0];


   w0=Vd(1:3,1);
   v0=Vd(4:6,1);
   CV=Cross(w0',v0')';
   Ard=[As(1,1); As(2,1); As(3,1);As(4,1)-CV(1,1);As(5,1)-
CV(2,1);As(6,1)-CV(3,1)];
```

```
   % The following statement compute the reverse analysis of the
manipulator    % for the required position at time t0+delta.
   [set1, set2, flag] = reverse_position_3R(a_, rd, thetad);
   if flag == 1
      error('The required position is outside the manipulator
workspace')
   else
      [Jd,waste] = Screw(a_,set1);
      % wd are the join angular velocities computed from the desired
      % velocity  state
      wd=Jd\Vd;
      Jwd=Jd*diag(wd);
      alphad=Jd\(Ard-Paccst(Jwd,3,1));

      % w are the join angular velocities and accelerations computed
from our
      % tracking algorithm
      w=2*(set1-set1c)/delta-wd;
      alpha=2*(set1-set1c)/delta^2;

      % Calculating Errors in Position, Orientation,
      % Linear and Angular velocity


      ta = [ ta [ t0 + delta ]];
      r0xa = [ r0xa [ rd(1,1) ]];
      r0ya = [ r0ya [ rd(2,1) ]];
      thetaa = [thetaa [thetad]];
      theta1a = [theta1a [set1(1,1)]];
      theta2a = [theta2a [set1(2,1)]];
      theta3a = [theta3a [set1(3,1)]];
      w01a=[w01a [ w(1,1)]];
      w12a=[w12a [ w(2,1)]];
      w23a=[w23a [ w(3,1)]];
      ew01a=[ew01a [w(1,1)-wd(1,1)]];
      ew12a=[ew12a [w(2,1)-wd(2,1)]];
      ew23a=[ew23a [w(3,1)-wd(3,1)]];
      alpha01a=[alpha01a [alpha(1,1)]];
      alpha12a=[alpha12a [alpha(2,1)]];
      alpha23a=[alpha23a [alpha(3,1)]];
      eal01a=[eal01a [alpha(1,1)-alphad(1,1)]];
      eal12a=[eal12a [alpha(2,1)-alphad(2,1)]];
      eal23a=[eal23a [alpha(3,1)-alphad(3,1)]];

      % Updating for the next iteration
      t0 = t0 + delta;
      set1c=set1;

   end


end
   % some plotting
   figure;
   plot(r0xa,r0ya), title ('Tracking of 3R manip. using inv. analysis
(pose, vel.)');
   legend('desired path');
   Draw3R(a_,theta1a,theta2a,theta3a);
   figure;
   plot(ta, w01a, ta, w01a+ew01a, ta, w12a, ta, w12a+ew12a,...
      ta, w23a, ta, w23a+ew23a), ...
      title('Actual and Desired Angular Velocities');
   legend('omega01','omega01_d','omega12','omega12_d',...
      'omega23','omega23_d');
   xlabel('time');
   figure;
   plot(ta, ew01a, ta, ew12a, ta, ew23a), ...
      title('Errors in Joint Angular Velocities');
   legend('error w01', 'error w12', 'error w23');
   xlabel('time');

   figure;
   plot(ta, alpha01a, ta, alpha01a+eal01a, ta, alpha12a, ...
      ta, alpha12a+eal12a, ta, alpha23a, ta, alpha23a+eal23a),...
      title('Actual and Desired Angular Accelerations');
   legend('alpha01','alpha01_d','alpha12','alpha12_d',...
      'alpha23','alpha23_d');
   xlabel('time');
   figure;
   plot(ta, eal01a, ta, eal12a, ta, eal23a), ...
      title('Errors in Joint Angular Accelerations');
   legend('error w01', 'error w12', 'error w23');
   xlabel('time');

   %figure;
   %plot(ta, w01a, ta, ta, w12a, ta, w23a),title('Joint Angular
Velocities');
   %figure;
   %plot(ta, alpha01a, ta, alpha12a, ta, alpha23a), title('Joint
Angular Accelerations');

end
```

```
% Movplat1c.m
% Program for path tracking of a parallel manipulator.
% Program uses the function reverse66.m to compute
% the Leg lengths, velocities, and accelerations


 t=0;

% The initial pose of the platform off the path
c0_=[0 ; 0 ; .5001];
cd0_=[0 ; 0 ; 0 ] ;
cdd0_=[0 ; 0 ; 0 ];
RMat0=[ cos(5*t^2+0.005) -sin(5*t^2+0.005) 0; sin(5*t^2+0.005)
cos(5*t^2+0.005) 0; 0 0 1];
wm0=[0 -10*t+0.05 0; 10*t+0.05 0 0; 0 0 0];
am0=[0  -10.0 0; 10.0 0 0; 0 0 0];


% The initial and final time for the tracking are
 t0=0.0;
 tf=4.0;
 delta=0.09; % TIME STEP


%Initialization of the arrays
ta=[];
Legsa=[];
VLegsa=[];
ALegsa=[];
Acoefa=[];
Posea=[];


%Performing the Inverse Position and Velocity analysis for the initial
position and Velocity.
[Legs0, VLegs0, ALegs0]= reverse66(c0_,RMat0, cd0_, wm0, cdd0_, am0);


%The loop starts here

while t<tf

t=t+delta;

% The path to be followed by the moving platform
c_=[0 ; 0.1 ; 0.5 ];
cd_=[ 0 ; 0 ; 0 ];
cdd_=[0 ; 0 ; 0 ];

RMat=[cos(5*t^2) -sin(5*t^2) 0; sin(5*t^2) cos(5*t^2) 0; 0 0  1];
wm=[0 -10*t 0; 10*t  0 0; 0 0 0];
am=[0  -10 0; 10 0 0; 0 0 0];

 [Legs, VLegs, ALegs]= reverse66(c_,RMat, cd_, wm, cdd_, am);
```

```
for i=1:6
  a3(i)=0.5*(  20*(Legs(i)-Legs0(i)) - delta*(8*VLegs(i)+12*VLegs0(i))
...
        + delta^2*(ALegs(i)+7*ALegs0(i)) )/delta^3;
  a4(i)=- (  15*(Legs(i)-Legs0(i)) - delta*(7*VLegs(i)+8*VLegs0(i)) ...
        + delta^2*(ALegs(i)+6*ALegs0(i)) )/delta^4;
  a5(i)=0.5*(  12*(Legs(i)-Legs0(i)) - 6*delta*(VLegs(i)+VLegs0(i)) ...
        + delta^2*(ALegs(i)+5*ALegs0(i)) )/delta^5;
end


%Updating the arrays
ta=[ta ;[t]];
Legsa=[Legsa ; [Legs]];
VLegsa=[VLegsa ;[VLegs]];
ALegsa=[ALegsa; [ALegs]];
Acoefa=[Acoefa ;[a3(1) a4(1) a5(1) a3(2) a4(2) a5(2) a3(3) a4(3) a5(3)
...
            a3(4) a4(4) a5(4) a3(5) a4(5) a5(5) a3(6) a4(6) a5(6)] ];
Posea=[Posea ;[RMat(1,:) RMat(2,:) RMat(3,:) c_']];

%Update the information
c0_=c_;
cd0_=cd_;
cdd0_=cdd_;
RMat0=RMat;
wm0=wm;
am0=am;

end


figure;
   axis equal;
   plot(ta, Legsa(:,1), '-', ta, Legsa(:,2), '--', ta, Legsa(:,3), ':',
ta, Legsa(:,4), '-.',...
   ta, Legsa(:,5), '+', ta, Legsa(:,6), '^'), ...
   title('Leg Displacements of the Platform)');
   legend('Leg1(l)','Leg2(q)','Leg3(m)','Leg4(r)',...
      'Leg5(n)','Leg6(p)');
   %axis([0.0, 1.0, 10, 30]);
   ylabel('in.');
   xlabel('time');

figure;
   axis equal;
   plot(ta, VLegsa(:,1), '-', ta, VLegsa(:,2), '--', ta, VLegsa(:,3),
':', ta, VLegsa(:,4), '-.', ...
   ta, VLegsa(:,5), '+', ta, VLegsa(:,6), '^'), ...
   title('Leg Velocities of the Platform');
   legend('Leg1(l)','Leg2(q)','Leg3(m)','Leg4(r)',...
      'Leg5(n)','Leg6(p)');
   %axis([0.0, 1.0, -15, 25]);
   ylabel('in./sec');
   xlabel('time');

figure;
```

```
   axis equal;
   plot(ta, ALegsa(:,1), '-', ta, ALegsa(:,2), '--', ta, ALegsa(:,3),
':', ta, ALegsa(:,4), '-.', ...
   ta, ALegsa(:,5), '+', ta, ALegsa(:,6), '^'), ...
   title('Leg Accelerations of the Platform');
   legend('Leg1(l)','Leg2(q)','Leg3(m)','Leg4(r)',...
      'Leg5(n)','Leg6(p)');
   %axis([0.0, 1.0, -30, 50]);
   ylabel('in./sec^2');
   xlabel('time');
```

```
% reverse66.m
% previous: reverse_position66.m (pos & vel)
%
% This function computes the leg (connector lengths) of
% the Special 6-6 parallel platform
% the platform is parametrized using
%   SIDE: determines the side of the platform (equi. triangle)
%   bscale: scales the distance of the pivot point as a fraction of
SIDE
%   tsidescale: scales the side of the top platform
%   tsacle: scales the distance of the pivot point on the top platform
%           as a fraction of (tsidesacle*SIDE)
%   hscale: determines the height of the top platform w.r.t base
%         (  again, as a fraction of SIDE)
%
% The INPUTS are the center of the top platform (c_) and the rotation
% Matrix of the top platfom RMat. Also cd_ is the velocity of the
center of top plat
% and wm is the angular velocity of top plat (it is a skew symmetric
matrix)
% AND the size of the platform in terms of the params. shown above
% as a 5x1 vector.
% The OUTPUT vectors are the 6 leg lengths and the velocities of the 6
legs

function [Legs, VLegs, ALegs] = reverse_position_66(c_, RMat, cd_, wm,
cdd_, am, size)
if nargin < 6
  error ('6-6 inv. pos. analysis requires 6 or 7 inputs'), end
if nargin > 7
  error ('6-6 inv. pos. analysis requires 6 or 7 inputs'), end
if nargin == 6
   % Platform size
   size(1,1) = 30.0; %SIDE
   size(2,1) = 0.8;  %bscale
   size(3,1) = 1.0;  %tsidescale
   size(4,1) = 0.8;  %tscale
   size(5,1) = 1.0;  %hscale
end

SIDE=size(1,1);
bscale=size(2,1);
tsidescale=size(3,1);
tscale=size(4,1);
hscale=size(5,1);

% forming the transform matrix from c_ and RMat
% and from cd_ and wm

for i=1:3
  for j=1:3
    TMat(i,j)=RMat(i,j);
    TVMat(i,j)=wm(i,j);
    TVMataux(i,j)=wm(i,j);
    TAMat(i,j)=am(i,j);
  end
  TMat(i,4)=c_(i,1);
  TVMat(i,4)=cd_(i,1);
  TVMataux(i,4)=0;
  TAMat(i,4)=cdd_(i,1);
end;
TMat(4,:)=[0,0,0,1];
TVMat(4,:)=[0,0,0,1];
TVMataux(4,:)=[0,0,0,0];
TAMat(4,:)=[0,0,0,1];

% Defining the joint locations on the base platform (fixed)
p1=[SIDE/2; -SIDE/(2*sqrt(3)); 0;1];
p2=[(1-bscale)*SIDE/2; (bscale-(1/3))*SIDE*sqrt(3)/2;0;1];
p3=[0;SIDE/sqrt(3);0;1];
p4=[-bscale*SIDE/2; -(bscale-(2/3))*SIDE*sqrt(3)/2; 0; 1];
p5=[-SIDE/2; -SIDE/(2*sqrt(3));0;1];
p6=[SIDE*(bscale-0.5); -SIDE*sqrt(3)/6; 0 ; 1];
pMat=[p1 p2 p3 p4 p5 p6];
%pMat

% Defining the joint locations on the top platform w.r.t to the
% center of the top platform
tSIDE=tsidescale*SIDE;
pt1=[tscale*tSIDE/2; (tscale-(2/3))*tSIDE*sqrt(3)/2;0;1];
pt2=[tSIDE/2; tSIDE/(2*sqrt(3));0;1];
pt3=[-tSIDE*(tscale-0.5); tSIDE/(2*sqrt(3));0;1];
pt4=[-tSIDE/2; tSIDE*sqrt(3)/6;0;1];
pt5=[-(1-tscale)*tSIDE/2; -(tscale-(1/3))*tSIDE*sqrt(3)/2;0;1];
pt6=[0; -tSIDE/sqrt(3); 0; 1];
ptMat=[pt1 pt2 pt3 pt4 pt5 pt6];

% Finding the joint locations on the top platform w.r.t the
% base platform (fixed)
ptMatfix=TMat*ptMat;
%ptMatfix

VMat=ptMatfix-pMat;
VelMat=TVMat*ptMatfix;
VelMataux=VelMat(1:3,1:6);
AccMat=TAMat*ptMatfix+TVMataux*(TVMataux*ptMatfix);

for i=1:6
  Legs(i)=sqrt(VMat(:,i)'*VMat(:,i));
end
Legmat=[Legs; Legs;  Legs; Legs];

uVMat=VMat./Legmat;

for i=1:6
  VLegs(i)=uVMat(:,i)'*VelMat(:,i);
  ALegs(i)=uVMat(:,i)'*AccMat(:,i)+(norm(VelMataux(:,i))^2/Legs(i))-...
          VLegs(i)^2/Legs(i);
end
```

```
% Movplat2b.m
% Program for path tracking of a parallel manipulator.
% Program determines the forces in each leg also
% REQUIRES THE FUNCTION "reverse66f.m"
% Implementing a constant acceleration motion (translation) with
% the Weight of the top platform ignored W = 0.

t=0;

% The initial pose of the platform off the path

c0_=[0.2 ; 0.4 ; 0.62 + 0.06*t^2];
cd0_=[0 ; 0 ; 0.06*2*t+.01] ;
cdd0_=[0 ; 0 ; 0.06*2+.001 ];
RMat0=[ 1 0 0; 0 1 0; 0 0 1];
wm0=[0 0 0; 0 0 0; 0 0 0];
am0=[0 0 0; 0 0 0; 0 0 0];

% The initial and final time for the tracking are
t0=0.0;
tf=0.50;
delta=0.02; % TIME STEP

%Initialization of the arrays
ta=[];
Legsa=[];
VLegsa=[];
ALegsa=[];
Acoefa=[];
Posea=[];
FLegsa=[];

%Performing the Inverse Position and Velocity analysis for the initial
position and Velocity.
[Legs0, VLegs0, ALegs0, FLegs0]= reverse66f(c0_,RMat0, cd0_, wm0,
cdd0_, am0);

%The loop starts here

while t<tf

t=t+delta;

% The path to be followed by the moving platform
c_=[0.2 ; 0.4 ; 0.6 + 0.06*t^2];
cd_=[ 0 ; 0 ; 0.06*2*t];
cdd_=[0 ; 0 ; 0.06*2];

RMat=[1 0 0; 0 1 0; 0 0 1];
wm=[0 0 0; 0 0 0; 0  0 0];
am=[0 0 0; 0 0 0; 0 0 0];

[Legs, VLegs, ALegs, FLegs]= reverse66f(c_,RMat, cd_, wm, cdd_, am);

for i=1:6
  a3(i)=0.5*(  20*(Legs(i)-Legs0(i)) - delta*(8*VLegs(i)+12*VLegs0(i))
...
```

```
        + delta^2*(ALegs(i)+7*ALegs0(i)) )/delta^3;
  a4(i)=- (  15*(Legs(i)-Legs0(i)) - delta*(7*VLegs(i)+8*VLegs0(i)) ...
        + delta^2*(ALegs(i)+6*ALegs0(i)) )/delta^4;
  a5(i)=0.5*(  12*(Legs(i)-Legs0(i)) - 6*delta*(VLegs(i)+VLegs0(i)) ...
        + delta^2*(ALegs(i)+5*ALegs0(i)) )/delta^5;
end

%Updating the arrays
ta=[ta ;[t]];
Legsa=[Legsa ; [Legs]];
VLegsa=[VLegsa ;[VLegs]];
ALegsa=[ALegsa; [ALegs]];
Acoefa=[Acoefa ;[a3(1) a4(1) a5(1) a3(2) a4(2) a5(2) a3(3) a4(3) a5(3)
...
            a3(4) a4(4) a5(4) a3(5) a4(5) a5(5) a3(6) a4(6) a5(6)] ];
Posea=[Posea ;[RMat(1,:) RMat(2,:) RMat(3,:) c_']];
FLegsa=[FLegsa; [FLegs']];

%Update the information
c0_=c_;
cd0_=cd_;
cdd0_=cdd_;
RMat0=RMat;
wm0=wm;
am0=am;

end

% some plotting

figure;
   axis equal;
   plot(ta, Legsa(:,1), '-', ta, Legsa(:,2), '--', ta, Legsa(:,3), ':',
ta, Legsa(:,4), '-.', ...
   ta, Legsa(:,5), 'v', ta, Legsa(:,6), '^'), ...
   title('Leg Displacements of the Platform');
   legend('Leg1(l)','Leg2(q)','Leg3(m)','Leg4(r)',...
      'Leg5(n)','Leg6(p)');
   %axis([0.0, 1.0, 10, 30]);
   ylabel('m');
   xlabel('time');

figure;
   axis equal;
   plot(ta, VLegsa(:,1), '-', ta, VLegsa(:,2), '--', ta, VLegsa(:,3),
':', ta, VLegsa(:,4), '-.',...
   ta, VLegsa(:,5), 'v', ta, VLegsa(:,6), '^'), ...
   title('Leg Velocities of the Platform');
   legend('Leg1(l)','Leg2(q)','Leg3(m)','Leg4(r)',...
      'Leg5(n)','Leg6(p)');
   %axis([0.0, 1.0, -15, 25]);
   xlabel('time');
   ylabel('m/sec');

figure;
   axis equal;
   plot(ta, ALegsa(:,1), '-', ta, ALegsa(:,2), '--', ta, ALegsa(:,3),
':', ta, ALegsa(:,4), '-.',...
   ta, ALegsa(:,5), 'v', ta, ALegsa(:,6), '^'), ...
   title('Leg Accelerations of the Platform');
   legend('Leg1(l)','Leg2(q)','Leg3(m)','Leg4(r)',...
      'Leg5(n)','Leg6(p)');
   %axis([0.0, 1.0, -30, 50]);
   xlabel('time');
   ylabel('m/sec^2');
 figure;
    axis equal;
    plot(ta, FLegsa(:,1), '-', ta, FLegsa(:,2), '--', ta, FLegsa(:,3),
':', ta, FLegsa(:,4), '-.',...
    ta, FLegsa(:,5), 'v', ta, FLegsa(:,6), '^'), ...
    title('Leg Forces of the Platform');
    legend('Leg1(l)','Leg2(q)','Leg3(m)','Leg4(r)',...
      'Leg5(n)','Leg6(p)');
    %axis([0.0, 1.0, -30, 50]);
    xlabel('time');
    ylabel('N');
```

```
% Movplat2d.m
% Program for path tracking of a parallel manipulator.
% Program determines the forces in each leg also
% REQUIRES THE FUNCTION "reverse66f.m"
% Implementing a rotation around c (about z), no translation,
% and constant angular acceleration
% the Weight of the top platform ignored W = 0.
% IMPORTANT ***** change Weight in "reverse66f.m"

t=0;

% The initial pose of the platform off the path

c0_=[0 ; 0 ; .5001];
cd0_=[0 ; 0 ; 0 ] ;
cdd0_=[0 ; 0 ; 0 ];
RMat0=[ cos(5*t^2+0.005) -sin(5*t^2+0.005) 0; sin(5*t^2+0.005)
cos(5*t^2+0.005) 0; 0 0 1];
wm0=[0 -10*t+0.05 0; 10*t+0.05 0 0; 0 0 0];
am0=[0  -10.0 0; 10.0 0 0; 0 0 0];

% The initial and final time for the tracking are
t0=0.0;
tf=0.50;
delta=0.02; % TIME STEP

%Initialization of the arrays
ta=[];
Legsa=[];
VLegsa=[];
ALegsa=[];
Acoefa=[];
Posea=[];
FLegsa=[];

%Performing the Inverse Position and Velocity analysis for the initial
position and Velocity.
[Legs0, VLegs0, ALegs0, FLegs0]= reverse66f(c0_,RMat0, cd0_, wm0,
cdd0_, am0);

%The loop starts here

while t<tf

t=t+delta;

% The path to be followed by the moving platform
c_=[0 ; 0.1 ; 0.5 ];
cd_=[ 0 ; 0 ; 0 ];
cdd_=[0 ; 0 ; 0 ];

RMat=[cos(5*t^2) -sin(5*t^2) 0; sin(5*t^2) cos(5*t^2) 0; 0 0  1];
wm=[0 -10*t 0; 10*t  0 0; 0 0 0];
am=[0  -10 0; 10 0 0; 0 0 0];

[Legs, VLegs, ALegs, FLegs]= reverse66f(c_,RMat, cd_, wm, cdd_, am);
```

```
for i=1:6
  a3(i)=0.5*(  20*(Legs(i)-Legs0(i)) - delta*(8*VLegs(i)+12*VLegs0(i))
...
        + delta^2*(ALegs(i)+7*ALegs0(i)) )/delta^3;
  a4(i)=- (  15*(Legs(i)-Legs0(i)) - delta*(7*VLegs(i)+8*VLegs0(i)) ...
        + delta^2*(ALegs(i)+6*ALegs0(i)) )/delta^4;
  a5(i)=0.5*(  12*(Legs(i)-Legs0(i)) - 6*delta*(VLegs(i)+VLegs0(i)) ...
        + delta^2*(ALegs(i)+5*ALegs0(i)) )/delta^5;
end

%Updating the arrays
ta=[ta ;[t]];
Legsa=[Legsa ; [Legs]];
VLegsa=[VLegsa ;[VLegs]];
ALegsa=[ALegsa; [ALegs]];
Acoefa=[Acoefa ;[a3(1) a4(1) a5(1) a3(2) a4(2) a5(2) a3(3) a4(3) a5(3)
...
            a3(4) a4(4) a5(4) a3(5) a4(5) a5(5) a3(6) a4(6) a5(6)] ];
Posea=[Posea ;[RMat(1,:) RMat(2,:) RMat(3,:) c_']];
FLegsa=[FLegsa; [FLegs']];

%Update the information
c0_=c_;
cd0_=cd_;
cdd0_=cdd_;
RMat0=RMat;
wm0=wm;
am0=am;

end

% some plotting

figure;
   axis equal;
   plot(ta, Legsa(:,1), '-', ta, Legsa(:,2), '--', ta, Legsa(:,3), ':',
ta, Legsa(:,4), '-.', ...
   ta, Legsa(:,5), 'v', ta, Legsa(:,6), '^'), ...
   title('Leg Displacements of the Platform');
   legend('Leg1(l)','Leg2(q)','Leg3(m)','Leg4(r)',...
      'Leg5(n)','Leg6(p)');
   %axis([0.0, 1.0, 10, 30]);
   ylabel('m');
   xlabel('time');

figure;
   axis equal;
   plot(ta, VLegsa(:,1), '-', ta, VLegsa(:,2), '--', ta, VLegsa(:,3),
':', ta, VLegsa(:,4), '-.',...
   ta, VLegsa(:,5), 'v', ta, VLegsa(:,6), '^'), ...
   title('Leg Velocities of the Platform');
   legend('Leg1(l)','Leg2(q)','Leg3(m)','Leg4(r)',...
      'Leg5(n)','Leg6(p)');
   %axis([0.0, 1.0, -15, 25]);
   xlabel('time');
   ylabel('m/sec');
```

```
figure;
   axis equal;
   plot(ta, ALegsa(:,1), '-', ta, ALegsa(:,2), '--', ta, ALegsa(:,3),
':', ta, ALegsa(:,4), '-.',...
   ta, ALegsa(:,5), 'v', ta, ALegsa(:,6), '^'), ...
   title('Leg Accelerations of the Platform');
   legend('Leg1(l)','Leg2(q)','Leg3(m)','Leg4(r)',...
      'Leg5(n)','Leg6(p)');
   %axis([0.0, 1.0, -30, 50]);
   xlabel('time');
   ylabel('m/sec^2');

 figure;
    axis equal;
    plot(ta, FLegsa(:,1), '-', ta, FLegsa(:,2), '--', ta, FLegsa(:,3),
':', ta, FLegsa(:,4), '-.',...
    ta, FLegsa(:,5), 'v', ta, FLegsa(:,6), '^'), ...
    title('Leg Forces of the Platform');
    legend('Leg1(l)','Leg2(q)','Leg3(m)','Leg4(r)',...
       'Leg5(n)','Leg6(p)');
    %axis([0.0, 1.0, -30, 50]);
    xlabel('time');
    ylabel('N');
```

```
% reverse66f.m
% previous: reverse66.m (pos, vel and accel)
%
% This function computes the leg (connector lengths) of
% the Special 6-6 parallel platform ALSO computes the
% leg velocities, leg accelerations, and LEG FORCES
% the platform is parametrized using
%   SIDE: determines the side of the platform (equi. triangle)
%   bscale: scales the distance of the pivot point as a fraction of
SIDE
%   tsidescale: scales the side of the top platform
%   tsacle: scales the distance of the pivot point on the top platform
%           as a fraction of (tsidesacle*SIDE)
%   hscale: determines the height of the top platform w.r.t base
%         (  again, as a fraction of SIDE)
%
% The INPUTS are the center of the top platform (c_) and the rotation
% Matrix of the top platfom RMat. Also cd_ is the velocity of the
center of top plat
% and wm is the angular velocity of top plat (it is a skew symmetric
matrix)
% AND the size of the platform in terms of the params. shown above
% as a 5x1 vector.
% The OUTPUT vectors are the 6 leg lengths and the velocities of the 6
legs
% The acceleration and forces in each of the legs

function [Legs, VLegs, ALegs, FLegs] = reverse_position_66(c_, RMat,
cd_, wm, cdd_, am, size)
if nargin < 6
  error ('6-6 inv. pos. analysis requires 6 or 7 inputs'), end
if nargin > 7
  error ('6-6 inv. pos. analysis requires 6 or 7 inputs'), end
I2M=0.0254; % in to meter
if nargin == 6
   % Platform size
   size(1,1) = 30.0*I2M; %SIDE
   size(2,1) = 0.8;  %bscale
   size(3,1) = 1.0;  %tsidescale
   size(4,1) = 0.8;  %tscale
   size(5,1) = 1.0;  %hscale
end

% Defining the Inertia Matrix (Mass in kgm, and Inertia in Kgm-m^2)
% For explanation see Rico notes

I=[0 0 0 79.5928 0 0; 0 0 0 0 79.5928 0; 0 0 0 0 0 79.5928; 3.39146 0 0
0 0 0; ...
   0 3.39167 0 0 0 0; 0 0 6.590532 0 0 0];

% Computing the Velocity and Reduced Acceleration States
V=[wm(3,2) ; wm(1,3); wm(2,1); cd_(1,1); cd_(2,1); cd_(3,1)];

% Dual part of the Reduced accel vector a0-w X v
DAR=cdd_-wm*cd_;
AR=[am(3,2) ; am(1,3); am(2,1); DAR(1,1); DAR(2,1); DAR(3,1)];
```

```
% Computing the Inertia Forces (weight for scr's platform)
Weight=[0; 0; 0; 0 ; 0; 0]; % W(-k) = -795.928

% Wrench on the top platform w.r.t center of moving platform (moving
ccs)
Wc=I*AR+Lie(V,I*V)-Weight ;

%This is an skew matrix for computing the cross product of rc/C.
Mc=[0 -c_(3,1) c_(2,1); c_(3,1) 0 -c_(1,1); -c_(2,1) c_(1,1) 0];

% Calculating  MC = Mc + r X F
Temp=Wc(4:6,1)+Mc*Wc(1:3,1);

% Wrench on the top platform w.r.t the fixed coordinate system
% by augmenting the F from Wc and MC from Temp(moment component)
W=[Wc(1:3,1); Temp(1:3,1)];

SIDE=size(1,1);
bscale=size(2,1);
tsidescale=size(3,1);
tscale=size(4,1);
hscale=size(5,1);

% forming the transform matrix from c_ and RMat
% and from cd_ and wm

for i=1:3
  for j=1:3
    TMat(i,j)=RMat(i,j);
    TVMat(i,j)=wm(i,j);
    TVMataux(i,j)=wm(i,j);
    TAMat(i,j)=am(i,j);
  end
  TMat(i,4)=c_(i,1);
  TVMat(i,4)=cd_(i,1);
  TVMataux(i,4)=0;
  TAMat(i,4)=cdd_(i,1);
end;
TMat(4,:)=[0,0,0,1];
TVMat(4,:)=[0,0,0,1];
TVMataux(4,:)=[0,0,0,0];
TAMat(4,:)=[0,0,0,1];

% Defining the joint locations on the base platform (fixed)
p1=[SIDE/2; -SIDE/(2*sqrt(3)); 0;1];
p2=[(1-bscale)*SIDE/2; (bscale-(1/3))*SIDE*sqrt(3)/2;0;1];
p3=[0;SIDE/sqrt(3);0;1];
p4=[-bscale*SIDE/2; -(bscale-(2/3))*SIDE*sqrt(3)/2; 0; 1];
p5=[-SIDE/2; -SIDE/(2*sqrt(3));0;1];
p6=[SIDE*(bscale-0.5); -SIDE*sqrt(3)/6; 0 ; 1];
pMat=[p1 p2 p3 p4 p5 p6];
%pMat

% Defining the joint locations on the top platform w.r.t to the
% center of the top platform
tSIDE=tsidescale*SIDE;
pt1=[tscale*tSIDE/2; (tscale-(2/3))*tSIDE*sqrt(3)/2;0;1];
```

```
pt2=[tSIDE/2; tSIDE/(2*sqrt(3));0;1];
pt3=[-tSIDE*(tscale-0.5); tSIDE/(2*sqrt(3));0;1];
pt4=[-tSIDE/2; tSIDE*sqrt(3)/6;0;1];
pt5=[-(1-tscale)*tSIDE/2; -(tscale-(1/3))*tSIDE*sqrt(3)/2;0;1];
pt6=[0; -tSIDE/sqrt(3); 0; 1];
ptMat=[pt1 pt2 pt3 pt4 pt5 pt6];

% Finding the joint locations on the top platform w.r.t to the center
of
% the top(moving) platform, expressed in the fixed coordinate system.
ptMatfix=TMat*ptMat;
%ptMatfix

VMat=ptMatfix-pMat;
VelMat=TVMat*ptMatfix;
VelMataux=VelMat(1:3,1:6);
AccMat=TAMat*ptMatfix+TVMataux*(TVMataux*ptMatfix);

for i=1:6
  Legs(i)=sqrt(VMat(:,i)'*VMat(:,i));
end
Legmat=[Legs; Legs;  Legs; Legs];
%VMat
%Legmat
%VelMat

uVMat=VMat./Legmat;
%uVMat

for i=1:6
  JL(:,i)=Cross((c_+ ptMatfix(1:3,i))',uVMat(1:3,i)')'; % Dual part of
the screw
  VLegs(i)=uVMat(:,i)'*VelMat(:,i);
  ALegs(i)=uVMat(:,i)'*AccMat(:,i)+(norm(VelMataux(:,i))^2/Legs(i))-...
          VLegs(i)^2/Legs(i);
end

J=[uVMat(1:3,:); JL]; % The Jacobian Matrix

%Solve the force analysis
FLegs=J\W; % The force is in Newtons
```

```
% main54cPID1.m
% July 2000
% An example for a 3R planar serial chain to study
% the proposed path tracking methodology
% using initial position and final position and velocity data
% Program requires the following functions:
% / Screw / Draw3R
% Inclues a PID Controller for a 3R Manipulator
%  Previous: main54c.m
%  Last Update: June 15, 2001


D2R = pi/180.; % Degrees to Radians conversion factor
%a_= [ 1 ; 1 ; 1]; % Constant data, link lengths
a_= [ .3 ; .3 ;.15];
r = 0.1; % r is a constant required by the end-effector path.

% The initial and final time, and the time step are given below
t0 = 0.0;
tf=2.0;
delta = 0.01 ;

% MOTOR Data
K = 1; % Gain for the system (motor)
tau = [ 0.005753 ; 0.00063 ; 0.0017 ]; % Motor time constants
% Kp = [ .75 ; 0.5 ; .426 ]; % Proportional gains (from sysmodelx.m)
% Kd = [ .0002 ;  0.000166 ; .0001076 ] ; % Derivative gains
% Ki = [ 285.8 ; 199.84 ; 200.33 ] ; % Integral gains
Kp = [ 1.077 ; 0.6156 ; 1.0626 ]; % Proportional gains (from
sysmodelx.m)
Kd = [ .000404 ;  0.000166 ; .0001076 ] ; % Derivative gains
Ki = [ 71.8 ; 56.84 ; 262.33 ] ; % Integral gains


D = 0.03; % Delay in seconds
% variables
deltac = delta/10; % Determines the speed at which the PID control
loop will run
% Constant for the DISCRETE CONTROLLER
    k0 = Kp + Ki*deltac + Kd/deltac;
    k1 = -Kp - (2*Kd/deltac);
    k2 = Kd/deltac;
for i=1:1:3
    e1(i)=0;
    e2(i)=0;
    e3(i)=0;
end
for i=1:1:(2+round(D/delta)) %
    x1(i)=0;
    x2(i)=0;
    x3(i)=0;
end
for i=1:1:2
    wa1(i)=0;
    wa2(i)=0;
    wa3(i)=0;
end
```

```
%Current Position Data:  initial position before starting
rc = [0.5995 ; 0.01 ; 0];
thetac = 0;
[set1c, set2c, flagc] = reverse_position_3R(a_, rc, thetac);
if flagc == 1
   error('The initial position is outside the manipulator workspace,
there is no point in completing the problem')
else

% Initializing all the vector arrays
ta = [];
r0xa = [];
r0ya =[];
thetaa=[];
theta1a=[];  theta2a=[];   theta3a=[];
w01a=[];     w12a=[];      w23a=[];
alpha01a=[]; alpha12a=[];  alpha23a=[];
ew01a=[];    ew12a=[];     ew23a=[];
eal01a=[];   eal12a=[];    eal23a=[];
w01f=[];  w12f=[];  w23f=[];
Wa1=[];  Wa2=[];  Wa3=[];
errorw1=[]; errorw2=[];  errorw3=[];



   % The Loop starts over here for the duration tf
   while t0<tf

   %The desired motion and velocity of the end-effector is given below
   rd = [0.5+r*cos((t0+delta)); r*sin((t0+delta)); 0];
   thetad = 0.0;
   Vd=[0;0;0;-r*sin(t0+delta);r*cos(t0+delta); 0];
   As=[0;0;0;-r*cos(t0+delta);-r*sin(t0+delta); 0];


   w0=Vd(1:3,1);
   v0=Vd(4:6,1);
   CV=Cross(w0',v0')';
   Ard=[As(1,1); As(2,1); As(3,1);As(4,1)-CV(1,1);As(5,1)-
CV(2,1);As(6,1)-CV(3,1)];


   % The following statement compute the reverse analysis of the
manipulator
   % for the required position at time t0+delta.
   [set1, set2, flag] = reverse_position_3R(a_, rd, thetad);
   if flag == 1
      error('The required position is outside the manipulator
workspace')
   else
      [Jd,waste] = Screw(a_,set1);
      % wd are the join angular velocities computed from the desired
      % velocity  state
      wd=Jd\Vd;
      Jwd=Jd*diag(wd);
      alphad=Jd\(Ard-Paccst(Jwd,3,1));
      % w are the join angular velocities and accelerations computed
from our
      % tracking algorithm
      w=2*(set1-set1c)/delta-wd;
      %alpha=2*(set1-set1c)/delta^2;
      alpha=(2/delta)*(wd-((set1-set1c)/delta)) ;

      % PID CONTROLLER WITH THE PLANT

      tcl=0;
      while ( tcl < delta )

          e1(1) = w(1,1) - wa1(1);
          e2(1) = w(2,1) - wa2(1);
          e3(1) = w(3,1) - wa3(1);

          x1(1) = x1(2) + k0(1,1)*e1(1) + k1(1,1)*e1(2) +
k2(1,1)*e1(3);
          x2(1) = x2(2) + k0(2,1)*e2(1) + k1(2,1)*e2(2) +
k2(2,1)*e2(3);
          x3(1) = x3(2) + k0(3,1)*e3(1) + k1(3,1)*e3(2) +
k2(3,1)*e3(3);

          wa1(1) = exp(-deltac/tau(1,1))*wa1(2) + K*(1 - exp(-
deltac/tau(1,1)))*x1(1+round(D/delta)) ;
          wa2(1) = exp(-deltac/tau(2,1))*wa2(2) + K*(1 - exp(-
deltac/tau(2,1)))*x2(1+round(D/delta)) ;
          wa3(1) = exp(-deltac/tau(3,1))*wa3(2) + K*(1 - exp(-
deltac/tau(3,1)))*x3(1+round(D/delta)) ;

          wa1(2) = wa1(1);
          wa2(2) = wa2(1);
          wa3(2) = wa3(1);
          e1(3) = e1(2);          e1(2) = e1(1);
          e2(3) = e2(2);          e2(2) = e2(1);
          e3(3) = e3(2);          e3(2) = e3(1);
          for j = length(x1):-1:2
              x1(j) = x1(j-1);
              x2(j) = x2(j-1);
              x3(j) = x3(j-1);
          end

          tcl = tcl + deltac;
      end

      % Array of actual velocities (from the control loop)
      Wa1 = [ Wa1 [ wa1(1) ]] ;
      Wa2 = [ Wa2 [ wa2(1) ]] ;
      Wa3 = [ Wa3 [ wa3(1) ]] ;
      % Array of velocity errors (between actual ang vel. 'wa' and
command velocity 'w')
      errorw1 = [ errorw1 [ wa1(1) - w(1,1) ]];
      errorw2 = [ errorw2 [ wa2(1) - w(2,1) ]];
      errorw3 = [ errorw3 [ wa3(1) - w(3,1) ]];

      wf=w+alpha*delta;
      % Calculating Errors in Position, Orientation,
      % Linear and Angular velocity

      ta = [ ta [ t0 + delta ]];
      r0xa = [ r0xa [ rd(1,1) ]];
      r0ya = [ r0ya [ rd(2,1) ]];
      thetaa = [thetaa [thetad]];
      theta1a = [theta1a [set1(1,1)]];
      theta2a = [theta2a [set1(2,1)]];
      theta3a = [theta3a [set1(3,1)]];
      w01a=[w01a [ w(1,1)]];
      w12a=[w12a [ w(2,1)]];
      w23a=[w23a [ w(3,1)]];

      w01f=[w01f [wf(1,1)]];
      w12f=[w12f [wf(2,1)]];
      w23f=[w23f [wf(3,1)]];

      ew01a=[ew01a [wf(1,1)-wd(1,1)]];
      ew12a=[ew12a [wf(2,1)-wd(2,1)]];
      ew23a=[ew23a [wf(3,1)-wd(3,1)]];

      % Updating for the next iteration
      t0 = t0 + delta;
      set1c=set1;

   end

end

   figure;
   axis normal;

   plot(ta, w01a, '--', ta, Wa1, '-', ta, w12a, '-.', ta, Wa2, '-', ta,
w23a, ':' , ta, Wa3, '-' ), ...
      title('Actual and Command Angular Velocities');
   legend('omega01','omega01_c','omega12','omega12_c',
'omega23','omega23_c');
   xlabel('time');
   figure;
   plot(ta, errorw1, '-', ta, errorw2,  '--', ta, errorw3, ':'), ...
      title('Errors in Joint Angular Velocities');
   legend('error w1', 'error w2', 'error w3');
   xlabel('time');

   %figure;
   %plot(ta, w01a, ta, ta, w12a, ta, w23a),title('Joint Angular
Velocities');
   %figure;
   %plot(ta, alpha01a, ta, alpha12a, ta, alpha23a), title('Joint
Angular Accelerations');
```

LIST OF REFERENCES

Abb00 Abbasi, W. A., Ridgeway, S. C., Adsit, P. D., Crane, C. D., and Duffy, J., "Investigation of a Special 6-6 Parallel Platform for Contour Milling." *Trans. of the ASME, Journal of Manufacturing Science and Engineering*, Vol. 122, No. 1, February 2000, pp: 132-139.

Abe92 Abe, S. and Tsuchiya, T., "Robot Manipulator Path Control Based on Variable Speed Trajectory Planning." *Advanced Robotics*, Vol. 6, No. 1, 1992, pp: 1-13.

Ami96 Amin, S. and Ahtiwash, O. M., "Performance of Two Neuro Controllers for Robot Path Planning Control." *Proc. 1996 IEEE 22nd Intl. Conf. on Industrial Electronics, Control, and Instrumentation*, August 1996, Taipei, Taiwan, Vol. 3, pp: 1856-1861.

Bal00 Ball, R. S., *A Treatise on the Theory of Screws*, Cambridge University Press, 1900. Paperback edition, 1999.

Bol88 Bollinger, John G. and Duffie, Neil A., *Computer Control of Machines and Processes*, Addison-Wesley Publishing Company, 1988.

Cah96 Cahill, A. J., Kieffer, J. C., and James, M. R., "Robust Time-Optimal Trajectory Planning for Robot Manipulators." *IEEE Robotics and Automation Conference*, 1996.

Cha30 Chasles, M., "Note sur les proprietes generales du systeme de deux corps semblables entr'eux et places d'une maniere quelconque dans l'espace; et sur le deplacement fini ou infiniment petit d'un corps solide libre." *Ferussac, Bulletin des Sciences Mathematiques*, Vol. xiv., pp: 321-326, 1830.

Cra99 Crane, C. and Duffy, J., "Geometry of Mechanisms and Robots II," Course notes for EML 6282, Graduate Course, Dept. of Mechanical Engineering, University of Florida, 1999.

Du95 Du, S., Tang, J., and Yang, G., "A New Method of Discrete Trajectory Planning for Robot Manipulators." *International Journal of Robotics and Automation*, Vol. 10, No. 4, 1995, pp: 123-129.

Fla88 Flash, T. and Potts, R. B., "Discrete Trajectory Planning." *International Journal of Robotic Research*, Vol. 6, No. 1, 1988, pp: 48-57.

Gra93 Grasa, P., Volberg, O., and Polyakhov, N., "Direct Adaptive Trajectory Control of Robot Manipulators via Fuzzy Logic Algorithm." *Proc. 8th Intl. Conf. on Applications of AI in Engineering*, Toulouse, France, Vol. 2, 1993, pp: 561-569.

Gri93 Griffis, M. and Duffy, J., "Method and Apparatus for Controlling Geometrically Simple Parallel Mechanisms with Distinctive Connections." US Patent No. 5,179,525, January 12, 1993.

Hun78 Hunt, K. H., *Kinematic Geometry of Mechanisms*, Oxford University Press, 1978.

Kie96 Kieffer, J. C., Cahill, A. J., and James, M. R., "Closed-Loop Trajectory Generation for Robust Time-Optimal Path Tracking." *Proc. of the 1996 IEEE International Conf. On Robotics & Automation*." Minneapolis, MN, April 1996, pp: 1559-1565.

Kie97 Kieffer, J., Cahill, A. J., and James, M. R., "Robust and Accurate Time-Optimal Path-Tracking Control for Robot Manipulators." *IEEE Trans. Robotics and Automation*, Vol. 13, No. 6, Dec. 1997, 880-890.

Li98 Li, Q., Tso, S. K., and Zhang, W. J., "Control of Robot Manipulators Using a Dual-Model-Based Structure." *Proc. of 1998 ASME DETC*, September 13-16, Atlanta, GA.

Lin83 Lin, C. S., Chang, P. R., and Luh, J. Y. S., "Formulation and Optimization of Cubic Polynomial Joint Trajectories for Industrial Robots," *IEEE Trans. on Automatic Control*, Vol. AC-28, No. 12, 1983, pp:1006-1017.

Luh77 Luh, J. Y. S. and Walker, M. W., "Minimum-time Along the Path for a Mechanical Arm." *Proc. 16th Conf. Decision Contr.*, Dec. 1977, pp:755-759.

Luh81 Luh, J. Y. S. and Lin, C. S., "Optimum Path Planning for Mechanical Manipulators." *ASME Journal of Dynamical Systems, Measurement, and Control.*, Vol. 102, June 1981, pp:142-151.

Mar96 Martinez, J. M. R. and Duffy, J., "An Application of Screw Algebra to the Acceleration Analysis of Serial Chains," *Mechanism and Machine Theory*, Vol. 31, No. 4, pp: 445-457, 1996.

Mas99 Mason, P. A. C., Walchko, K., and Crane, C. D., "A MIMO Fuzzy Controller. for Tracking: Robot Control." *1999 Florida Conf. on Recent Advances in Robotics*, April 1999, Gainesville, FL. http://cimar.me.ufl.edu/FLA99/papers.html.

Moo79 Moore, R. E., *Methods and Applications of Interval Analysis*. Philadelphia, PA. SIAM Press, 1979.

Naj96 Najson, F. and Kreindler, E., "Robot Robust Path Tracking." *Dynamics and Control*, Vol. 6, 1996, pp: 333-359.

Oga97 Ogata, K., *Modern Control Engineering*, 3rd Edition, Prentice-Hall, 1997.

Par97 Park, S. and Park, C. H., "Model-based Path Planning and Tracking Control Using Neural Networks for a Robot Manipulator." *Conf. Proc. of the 1997 IEEE Intl. Conf. on Neural Networks*, Houston, TX, Vol. 3, June 1997, pp: 1761-1765.

Pau81 Paul, R. P. C., *Robot Manipulators: Mathematics, Programming, and Control*. Cambridge, MA. M.I.T Press, 1981.

Phi84 Phillips, J., *Freedom in Machinery. Vol. 1, Introducing Screw Theory*, Cambridge University Press, 1984.

Pia97 Piazzi, A., and Visioli, A., "An Interval Algorithm for Minimum-Jerk Trajectory Planning of Robot Manipulators." *Proc. Of the 36th Conf. On Decision & Control*, San Diego, CA, USA, December 1997.

Poi06 Poinsot, L., "Sur la composition des moments et la composition des aires" (1804). *Journal de l'Ecole Polytechnique*, Vol. vi. (13 ch.), pp: 182-205, 1806.

Rag93 Raghavan, M., "The Stewart Platform of General Geometry has 40 Configurations." *Trans. of the ASME, Journal of Mechanical Design*, Vol. 115, June 1993, pp: 272-282.

Rel Reliance Motor Control, Inc., *DC Motors, Speed Controls, Servo Systems: The Electro-Craft Engineering Handbook*, Reliance Motor Control Inc., Eden Prairie, Minnesota.

Ric99 Rico, J. M., Gallardo, J., and Duffy, J., "Screw Theory and Higher Order Kinematic Analysis of Open Serial and Closed Chains." *Mechanism and Machine Theory*, Vol. 34, pp: 559-586, 1999.

Shi85 Shin, Kang G. and McKay, Neil D., "Minimum-Time Control of Robotic Manipulators with Geometric Path Constraints." *IEEE Transactions on Automatic Control*, Vol. AC-30, No. 6, 1985, pp: 531-541.

Shi86a Shin, Kang G. and McKay, Neil D., "A Dynamic Programming Approach to Trajectory Planning of Robotic Manipulators." *IEEE Transactions on Automatic Control*, Vol. AC-31, No. 6, 1986, pp:491-500.

Shi86b Shin, Kang G. and McKay, Neil D., "Selection of Near-Minimum Time Geometric Paths for Robot Manipulators." *IEEE Transactions on Automatic Control*, Vol. AC-31, No. 6, 1986, pp: 501-511.

Shi87 Shin, K. J. and McKay, N. D., "Robust Trajectory Planning for Robotic Manipulators Under Payload Uncertainties." *IEEE Trans. Automatic Control*, Vol. AC-32, No. 12, 1987, pp: 1044-1054.

Slo87 Slotine, J. J. E. and Li, W., "On the Adaptive Control of Robot Manipulators." *International Journal of Robotics Research*, Vol. 6, 1987, pp: 49-59.

Ste65 Stewart, D., "A Platform with Six Degrees of Freedom," *Proc. Institution of Mechanical Engineers*, Vol. 180, Pt. 1, No. 15, 1965, pp: 371-386.

Tan88 Tan, H. H. and Potts, R. B., "Minimum Time Trajectory Planner for the Discrete Dynamic robot Model with Dynamic Constraints." *IEEE Journal of Robotics and Automation*, Vol. 4, No. 2, 1988, pp: 174-185.

Tsu90 Tsuchiya, T. and Egami, T., "Robot Path Control with Variable Speed by Preview Control and Adaptive Control." *IFAC 11th Triennial World Congress*, Tallinn, Estonia, USSR, 1990, pp: 121-126.

Wan93 Wang, S., Tsuchiya, T., and Hashimoto, Y., "Path Tracking Control of Robot Manipulators Utilizing Future Information of Desired Trajectory." *Trans. Of Japan Society of Mechanical Engineers*, Part C, Vol. 5, No. 564, Aug. 1993, pp: 2512-2518.

Wan94 Wang, S. and Tsuchiya, T., "Fuzzy Trajectory Planning and its Application to Robot Manipulator Path Control." *Advanced Robotics*, Vol. 8, No. 1, 1994, pp: 73-93.

Xia94 Xiangrong, X. and Xiangfeng, M., "A Method for Trajectory Planning of Robot Manipulators." *Modeling, Measurement & Control B:*, ASME Press, Vol. 54, No. 3, 1994, pp:33-36.

Zad69 Zadeh, L. A., "Fuzzy Sets." *Information Control*, Vol. 8, No. 3, 1965, pp: 338-353.

Zel95 Zelek, J. S., "Dynamic Path Planning." *Proc. of the IEEE International Conference on Systems, Man and Cybernetics*, Vol. 2, 1995, Piscataway, NJ, pp: 1285-1290.

BIOGRAPHICAL SKETCH

Waheed A. Abbasi was born on August 12, 1968 in the southern port-city of Karachi, Pakistan. To undertake his undergraduate studies, he moved to the Cornhusker state, where he received his Bachelor of Science degree in Mechanical Engineering from the University of Nebraska-Lincoln (UNL) in 1991. His interest in Robotics and his desire to pursue graduate study brought him to Gator country, where he joined the Center for Intelligent Machines and Robotics (CIMAR), Department of Mechanical Engineering, University of Florida (UF). He received his Master of Science degree in Mechanical Engineering with a minor in Industrial Engineering in December 1994. Afterwards, he continued his studies and work at CIMAR, being involved in a variety of projects as well as conducting research. Upon completion of his doctoral degree, he intends to work in the industry. Waheed is married to Uzma and they have a son, Bassil.

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

Carl D. Crane III, Chairman
Professor of Mechanical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

Joseph Duffy
Graduate Research Professor of
Mechanical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

John C. Ziegert
Professor of Mechanical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

John K. Schueller

John K. Schueller
Associate Professor of Mechanical
Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]
Michael C. Nechyba
Assistant Professor of Electrical and
Computer Engineering

This dissertation was submitted to the Graduate Faculty of the College of Engineering and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

August 2001

Pramod Khargonekar
Pramod P. Khargonekar
Dean, College of Engineering

Winfred M. Phillips
Dean, Graduate School

LD
1780
2001
.A122